PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H01J 17/30 | A2

(11) International Publication Number: WO 99/27556

(43) International Publication Date: 3 June 1999 (03.06.99)

(21) International Application Number: PCT/US98/24963

(22) International Filing Date: 20 November 1998 (20.11.98)

(30) Priority Data:

60/066,898	20 November 1997 (20.11.97)	US
60/109,095	19 November 1998 (19.11.98)	US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Applications

US	60/066,898 (CIP)
Filed on	20 November 1997 (20.11.97)
US	60/109,095 (CIP)
Filed on	19 November 1998 (19.11.98)

(71) Applicant *(for all designated States except US)*: XACCT TECHNOLOGIES, INC. [US/US]; Suite 105, 2855 Kifer Road, Santa Clara, CA 95051 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: WAGNER, Eran [US/US]; Apartement 217, 20677 Forge Way, Cupertino, CA 95014 (US). SCHWEITZER, Limor [IL/IL]; Achimeir Street 12/42, Ramat aviv gimel (IL). GIVOLY, Tal [IL/IL]; Barkan Street 2/1, 44288 Kfar-Saba (IL).

(74) Agent: RICHARDSON, Kent, R.; Wilson Sonsini Goodrich & Rosati, 650 Page Mill Road, Palo Alto, CA 94304-1050 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: NETWORK ACCOUNTING AND BILLING SYSTEM AND METHOD

(57) Abstract

In some embodiments, network traffic information is captured at network information sources. These sources provide detailed information about the network communications transactions at a network device. Importantly, different types of sources can provide different types of information. Gatherer devices gather the detailed information from the various information source devices and convert the information into standardized information. The gatherer devices can correlate the gathered information with account information for network transaction accounting. Manager devices manage the gatherer devices and store the gathered standardized information. The manager devices eliminate duplicate network information that may exist in the standardized information. The manager devices also consolidate the information. Importantly, the information stored by the manager devices represents the consolidated, account correlated, network transaction information that can be used for billing or network accounting. The system thereby provides a distributed network accounting and billing system.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

NETWORK ACCOUNTING AND BILLING SYSTEM AND METHOD

RELATED APPLICATIONS

This application relates to, and incorporates by reference, the United States patent application having serial number 60/066,898, entitled "Network Accounting and Billing System," having inventors Limor Schweitzer and Eran Wagner, filed November 20, 1997, and which has a common assignee. This application also relates to the United States patent application having serial number XX/XXX,XXX, entitled "Network Accounting and Billing System," having inventors Limor Schweitzer and Eran Wagner, filed November 19, 1998, and which has a common assignee.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

A. Field of the Invention

This invention relates to the field of computer networks. In particular, the invention relates to accounting and billing for services in a computer network.

B. Description of the Related Art

The low cost of Internet connectivity and a wide range of services are driving and more people onto the Internet, which is driving the deployment of TCP/IP networks. This process has led to a new market of client-server applications that enables the user to interact with other users and computer systems around the world. The use of these applications is consuming more and more Intranet and Internet bandwidth.

New applications such as "voice over IP (Internet Protocol)" and streaming audio and video require even more bandwidth and a different quality of service than email, or other less real-time applications. Also, the type quality of service can vary according to the needs of the user. For example, typically, businesses do not tolerate unavailable network services as easily as consumers. Internet Service Providers (ISPs) therefore would like to price their available bandwidth according to a user's needs. For example, flat monthly pricing may be the best billing model for consumers, but businesses may want to be billed according to their used bandwidth at particular qualities of service.

As ISPs continue to differentiate themselves by providing additional services, enterprise information technology managers will face similar problems to account for the escalating Intranet operating costs.

Therefore, ISPs and enterprise information technology managers will want to account for session logging, bandwidth usage, directory data and application session information from a variety of sources.

Due to the diversity of IP data sources (e.g., routers, hubs etc.), the need for effect tracking far exceeds the problems addressed by telephone companies. Telephone companies track information such as circuit usage so it can be correlated with account information. For example, businesses may use leased lines, consumers may have "Friends and Family" plans, cellular phones have different roamer fees according to the location of the user, etc. Typically, the phone company captures all of the data and uses batch processing to aggregate the information into specific user accounts. For example, all the long distance calls made during a billing period are typically correlated with the Friends and Family list for each phone account at the end of a billing period for that account. This requires a significant amount of computing power. However, this type of problem is significantly simpler than attempting to track and bill for every transaction in an IP network. Therefore, what is desired is a system that allows for accounting and billing of transactions on IP based networks.

The problem is even more difficult in IP network traffic because the information sources can exist and many different levels of the OSI network

model. throughout heterogeneous networks. Potential sources of information include packet use from routers, firewall authentication logging, email data, ISP session logging, and application layer use information. Therefore, what is desired is a system and method that track IP network usage information across multiple layers of the OSI network model.

SUMMARY OF THE INVENTION

A network accounting and billing system and method are described. In some embodiments, the system can access any network related information sources such as traffic statistics provided by routers and switching hubs as well as application server access logs. The information can be accumulated in a central database for creating auditing, accounting and billing reports. Alternatively, the information can be sent directly to other systems such as rating engines used in customer care and billing systems.

In one embodiment, network traffic information is captured at network information sources (examples of information sources include network devices). These sources provide detailed information about the network communications transactions at a network device. Importantly, different types of sources can provide different types of information. Gatherer devices gather the detailed information from the various information source devices and convert the information into standardized information. The gatherer devices can correlate the gathered information with account information for network transaction accounting. Manager devices manage the gatherer devices and store the gathered standardized information. The manager devices eliminate duplicate network information that may exist in the standardized information. The manager devices also consolidate the information. Importantly, the information stored by the manager devices represents the consolidated, account correlated,

network transaction information used for billing. In addition to account information, transaction information can be correlated to other information such as geography information (e.g., the location of an accessed server) and/or transaction routing information (as may be used in peering agreements between Internet Service Providers). The system thereby provides a distributed network accounting and billing system.

In some embodiments, the gatherer devices can access sources through proxy gateways, firewalls, and/or address translation barriers.

In some embodiments, the gatherer devices can correlate the information about a specific transaction with a particular account by accessing the transaction's source and/or destination information. The source and/or destination information is then correlated with account information from an account information database.

BRIEF DESCRIPTION OF THE FIGURES

The figures illustrate the invention by way of example. The invention is not meant to be limited to only those embodiments of shown in the Figures. The same reference in different figures indicates the same element is being used in those figures.

Figure 1 illustrates a system including one embodiment of the invention.

Figure 2 illustrates an example of the data distillation used in the system of Figure 1.

Figure 3 illustrates data enhancements used in the data distillation.

Figure 4A illustrates example field enhancements that can be included in the data enhancements.

Figure 4B illustrates the creation of an enhanced record.

Figure 5 illustrates an example record merge.

Figure 6 illustrates an example of an alternative embodiment of the system.

DETAILED DESCRIPTION

A. System Overview

One embodiment of the system includes a multi-source, multi-layer network usage metering and mediation solution that gives Network Service Providers (NSPs), including Internet Service Providers (ISPs) and enterprise network(Intranet) operators, the information needed to set the right-price for IP (Internet Protocol) services. With the system, the providers can generate accurate usage-based billing and implement usage-based charge-back models. The system derives IP session and transaction information, collected in real time, from a multitude of network elements. The system gathers, correlates, and transforms data from routers, switches, firewalls, authentication servers, LDAP, Web hosts, DNS, and other devices to create comprehensive usage and billing records.

The system transforms raw transaction data from network devices into useful billing records though policy-based filtering, aggregation, and merging. The result is a set of detail records (DRs). In some embodiments, the detail records are XaCCT Detail Records (XDRs™) available from XaCCT Technologies. DRs are somewhat similar in concept to the telephony industry's Call Detail Records (CDRs). Thus, DRs can be easily integrated with existing Customer Care and Billing (CCB) systems.

In addition to billing data, DRs enable NSPs to deploy new services based on documented usage trends, plan network resource provisioning, and audit service usage. The system provides a clear picture of user-level network service use by tracking a variety of metrics such as actual session Quality of Service (QoS),traffic routes, and end-user application transactions.

The system is based on a modular, distributed, highly scalable architecture capable of running on multiple platforms. Data collection and management is designed for efficiency to minimize impact on the network and system resources.

The system minimizes network impact by collecting and processing data close to its source. Modular architecture provides maximum configuration flexibility, and compatibility with multiple network information sources.

The system, or other embodiments, may have one or more of the following features.

Data collection can be from a wide range of network devices and services, spanning all layers of the network - from the physical layer to the application layer.

Real-time, policy-based filtering, aggregation, enhancement and merging creates accurate, detailed and comprehensive session detail records (DRs).

Real time correlation of data from various sources allows billing record enhancement.

Leverages existing investment through integration with any customer care & billing solution, reducing costs, minimizing risks and shortened time-to-market.

Non-intrusive operation eliminates any disruption of network elements or services.

Web-based user interface allows off-the-shelf browsers to access the system, on-demand, locally or remotely.

Carrier-class scalability allows expansion to fit an NSPs needs without costly reconfiguration.

Distributed filtering and aggregation eliminates system capacity bottlenecks.

Efficient, centralized system administration allows on-the-fly system reconfigurations and field upgrades.

Customized reporting with built-in report generation or an NSPs choice of off-the-shelf graphical reporting packages.

Comprehensive network security features allow secure communication between system components and multiple levels of restricted access.

B. System Details

The following describes the system 100 of Figure 1. The system 100 allows NSPs to account for and bill for IP network communications. The following paragraphs first list the elements of Figure 1, then describes those elements and then describes how the elements work together. Importantly, the

distributed data gathering, filtering and enhancements performed in the system 100 enables load distribution. Granular data can reside in the peripheries of the system 100, close to the information sources. This helps avoids reduce congestion in network bottlenecks but still allows the data to be accessible from a central location. In previous systems, all the network information flows to one location, making it very difficult to keep up with the massive record flows from the network devices and requiring huge databases.

The following lists the elements of Figure 1. Figure 1 includes a number of information source modules (ISMs) including an ISM 110, an ISM 120, an ISM 130, an ISM 136, an ISM 140, and an ISM 150. The system also includes a number of network devices, such as a proxy server 101, a DNS 102, a firewall 103, an LDAP 106, a CISCO NetFlow 104, and a RADIUS 105. The system also includes a number of gatherers, such as a gatherer 161, a gatherer 162, a gatherer 163, a gatherer 164, and a gatherer 165. The system of Figure 1 also includes a central event manager (CEM) 170 and a central database (repository) 175. The system also includes a user interface server 185 and a number terminals or clients 180.

This paragraph describes how the elements of Figure 1 are coupled. The various network devices represent devices coupled to an IP network such as the Internet. The network devices perform various functions, such as the proxy server 101 providing proxy service for a number of clients. Each network device is coupled to a corresponding ISM. For example, the proxy server 101 is

coupled to the ISM 110. The DNS 102 is coupled to the ISM 120. The firewall 103 is coupled to the ISM 130. The ISM 136 is coupled to the LDAP 106. The ISM 140 is coupled to the CISCO NetFlow 104. The ISM 150 is coupled to the RADIUS 105. Each gatherer is associated with at least one ISM. Thus, the gatherer 161 is associated with the ISM 110 and is therefore coupled to that ISM. The gatherer 162 is coupled to the ISM 120. The gatherer 163 is coupled to the ISM 130 and the ISM 136. The gatherer 164 is coupled to the ISM 140. The gatherer 165 is coupled to the ISM 150. The various gatherers are coupled to the CEM 170. The user interface server is coupled to the terminals 180 and the CEM 170.

The following paragraphs describe each of the various elements of Figure 1.

Network Devices

The network devices represent any devices that could be included in a network. (Throughout the description, a network device, unless specifically noted otherwise, also refers to an application server.) A network device represents a subset of information sources that can be used by the system 100. That is, the network devices are merely representative of the types of sources of information that could be accessed. Other devices such as on-line transaction processing databases can be accessed in other embodiments of the invention. Typically, the network devices keep logging and statistical information about their activity. A network information source can be the log file of a mail server,

the logging facility of a firewall, a traffics statistics table available on a router and accessible through SNMP, a database entry accessible through the Internet, an authentication server's query interface, etc. The network devices represent the information sources accessed by the ISMs.

Each type of network device can be accessing using a different method or protocols. Some generate logs while others are accessible via SNMP, others have proprietary APIs or use other protocols.

ISMs

The ISMs act as an interface between the gatherers and the network devices enabling the gatherers to collect data from the network devices. Thus, the ISMs represent modular, abstract interfaces that are designed to be platform-neutral. The information source modules act as interfaces or "translators", sending IP usage data, in real time, from the network devices to the gatherers. Each ISM is designed for a specific type of network data source. (In other embodiments, some ISM are generic in that they can extract information from multiple network devices). ISMs can be packaged separately, allowing NSPs to customize ISM configurations to meet the specific requirements of their network. For example, in the system of Figure 1, if the NSP did not have Cisco NetFlow devices, then the ISM 140 would not have to be included.

The ISMs can communicate with its corresponding network device using protocols and formats such as UDP/IP, TCP/IP, SNMP, telnet, file access, ODBC, native API, and others.

In some embodiments, the reliability of system 100 is enhanced through on-the-fly dynamic reconfiguration, allowing the NSP to add or remove modules without disrupting ongoing operations. In these embodiments, the CEM 170 can automatically update the ISMs.

The following ISMs are available in some embodiments of the invention.

- Categorizer - Classifies a session to a category according to user-defined Boolean expression.
- DNS (e.g. ISM 120) - Resolves host names and IP addresses.
- Generic Proxy Server (e.g., ISM 110) - Collects data from access logs in a common log format.
- Port / Protocol Resolution - Converts protocol/port information to account names and vice versa.
- CheckPoint FireWall-1 - Collects data from FireWall-1 accounting log and security log.
- Cisco IOS IP Accounting - Collects accounting data from a Cisco router using IOS IP accounting.
- Cisco NetFlow Switching - Collects session data from a Cisco router via NetFlow switching.
- NETRANET – Collects information from a standard network device.
- Netscape Proxy Server - Collects data from a Netscape Proxy Server.

- Microsoft Proxy Server - Collects data from a Microsoft Proxy Server.

ISMs can be synchronous, asynchronous or pipe.

The data from an asynchronous ISM is dynamic so that the asynchronous ISM reacts to the information and relays it to the associated gatherer without prompting from other information sources in the system 100. If the firewall 103 were a CheckPoint FireWall-1, then the ISM 130 would be an example of an asynchronous ISM. When a network session is initiated, the details are recorded by the FireWall-1 103. The corresponding ISM 130 receives the details and passes them on automatically to the gatherer 163.

Synchronous ISMs provide its information only when accessed by a gatherer. The ISM 120 is an example of a synchronous ISM. The DNS server 102 maintains information matching the IP addresses of host computers to their domain addresses. The ISM 120 accesses the DNS server 102 only when the ISM 120 receives a request from the gather 162. When the DNS server 102 returns a reply, the ISM 120 relays the reply information to the gatherer 162.

Pipe ISMs operate on record flows (batches of records received from information sources). Pipe ISMs process one or more enhancement flows the records as the flows arrive. The pipe ISM may initiate new record flows or may do other things such as generate alerts or provision network elements to provide or stop services. The pipe is implemented as an ISM to keep the internal coherency and logic of the architecture. (Record flows can terminate in a

database or in a pipe ISM. The pipe ISM can perform filtering and aggregation, send alarms, or act as a mediation system to provision network elements when some event occurs or some accumulated value is surpassed. Specifically, pipe ISMs can act to enable pre-payment systems to disable certain services such as a voice IP call, when the time limit is surpassed or amount of data is reached.)

The gatherers can include caches and buffers for storing information from the ISMs. The buffers allow the gatherers to compensate for situations where there is a loss of connection with the rest of the system 100. The cache sizes can be remotely configured. The cache minimizes the number of accesses to the Information Source.

ISM queries can be cached and parallelized. Caching of synchronous ISM queries provides for fast responses. Parallelizing queries allows for multiple queries to be processed at the same time.

Gatherers

The gatherers gather the information from the ISMs. In some embodiments, the gatherers are multi-threaded, lightweight, smart agents that run on non-dedicated hosts, as a normal user application on Windows NT or Unix, as a background process, or daemon. What is important though is that the gatherers can be any hardware and/or software that perform the functions of a gatherer.

The gatherers can be installed on the same network segment as the network device such as router and switch or on the application server itself. This placement of a gatherer minimizes the data traffic impact on the network.

The gatherers collect network session data from one or more ISMs. Session data can be sent to another gatherer for enhancement or to the CEM 170 for merging and storing in the central database 170. The gatherers can be deployed on an as needed basis for optimal scalability and flexibility.

The gatherers perform flexible, policy-based data aggregation. Importantly, the various types of ISMs provide different data and in different formats. The gatherers normalize the data by extracting the fields needed by the CEM 170 and filling in any fields that may be missing. Thus, the gatherers act as a distributed filtering and aggregation system. The distributed data filtering and aggregation eliminates capacity bottlenecks improving the scalability and efficiency of the system 100 by reducing the volume of data sent on the network to the CEM 170.

Aggregation can be done by accumulating groups of data record flows, generating a single data record for each group. That single record then includes the aggregated information. This reduces the flow of the data records.

Filtering means discarding any record that belongs to a group of unneeded data records. Data records are unneeded if they are known to be collected elsewhere. A policy framework enables the NSP to configure what to collect where.

Filtering and/or aggregation can be done at any point along a data enhancement (described below) so that aggregation schemes can be based on enhanced data records as they are accumulated. The filtering and/or aggregation points are treated by the system 100 as pipe ISMs which are flow termination and flow starting points (ie: like an asynchronous ISM on the starting end and like a database on the terminating end). Data enhancement paths and filtering and/or aggregation schemes can be based on accumulated parameters such as user identification information and a user's contract type.

As noted above, the PISM can be used in the context of filtering and/or aggregation. One or more record flows can terminate at the PISM and can be converted into one or more new record flows. Record flows are grouped based on matching rules that apply to some of the fields in the record flows, while others are accumulated or undergo some other operation such as "maximum" or "average". Once the groups of accumulated records have reached some threshold, new accumulated records are output. This can be used for example in order to achieve a business-hybrid filtering and aggregation data reduction by imposing the business rules or the usage-based products that are offered to the customer, onto the record flows as they are collected in real-time. This is done instead of previous system where the information is stored in a database and then database operations are performed in order to create bills or reports. The filtering and aggregation reduces the amount of data that is stored in the central

database 175 while not jeopardizing the granularity of data that is necessary in order to create creative usage-based products.

Typically, data collected from a single source does not contain all the information needed for billing and accounting, such as user name and organization. In such cases, the data is enhanced. By combining IP session data from multiple sources, such as authentication servers, DHCP and Domain Name servers, the gatherers create meaningful session records tailored to the NSP's specific requirements. In the example of Figure 1, the gatherer 161 can provide information to the gatherer 162 so that the source IP address for an Internet session from the proxy server 101 can be combined with the domain address from the DNS server 102.

The enhancement procedure can be triggered by an asynchronous ISM. The information from the asynchronous ISM is associated with field enhancements in the central database 175. A field enhancement defines how a field in the central database is filled from the source data obtained from the asynchronous ISM. Through the field enhancements, the missing parameters are added to a record using the data collected from one or more synchronous ISMs. Enhancements are described in detail below.

The gatherers can include caches and buffers for storing information from the ISMs. The buffers allow the gatherers to compensate for situations where there is a loss of connection with the rest of the system 100. The caches

can reduce the number of accesses to an information source. The buffer and/or cache sizes can be remotely configured.

Central Event Manager (CEM)

The Central Event Manager (CEM) 170 acts as the central nervous system of the system 100, providing centralized, efficient management and controls of the gatherers and the ISMs.

The CEM 170 can perform one or more of the following tasks:

- Coordinates, controls, and manages the data collection process. The CEM 170 coordinates the operation of the gatherers and manages the flow of data through the system 100 through the collection scheme defined in the system configuration. The latter includes the configuration of the gatherers, the ISMs, the network devices, the fields in the central database 175 (described below), and the enhancement procedures. Based on the collection scheme the CEM 170 determines the system 100's *computation flow* (the set of operations the system 100 must perform to obtain the desired information). The CEM 170 then controls all the gatherers, instructing them to perform, in a particular sequence, the operations defined in the computation flow. The CEM 170 receives the records collected by the gatherers and stores them in the central database 175. NSPs can configure the CEM 170 to *merge* duplicate records

before storing them in the central database 175. Record merging is described below.

- Performs clean-up and aging procedures in the database 175. The system 100 collects and stores large amounts of session information every day. The CEM 170 removes old data to free space for new data periodically. The NSP defines the expiration period for the removal of old records. The CEM 170 is responsible for coordinating the removal of records from the central database 175. The CEM 170 places a time stamp on every record when the record enters the central database 175 and deletes the record after the time period the NSP has defined elapses.
- Provides centralized system-wide upgrade, licensing, and data security. The NSP can perform version upgrades of the system 100 at the CEM 170. The gatherers can be automatically upgraded once a new version is installed on the host computer of the CEM 170. ISMs are also installed via the CEM 170 and exported to the gatherers. The CEM 170 maintains a list of licenses installed in the system and verifies periodically if the system is properly licensed. This feature lets the NSP centrally install and uninstall licenses. It also prevents unlicensed use of the system 100 and any of its components.
- Monitors the state of the gatherers and ISMs. The gatherers periodically communicate with the CEM 170. The CEM 170

continuously monitors the state of each gatherer and network devices in the system 100. The CEM 170 can be fault-tolerant, that is, it can recover from any system crash. It coordinates the recovery of the system 100 to its previous state.

In some embodiments, a key directory server is associated with the CEM 170. To transfer less data between the elements of the system 100, it is desirable that each piece of data to carry little descriptive data. For example, if IP address data is transferred between a gatherer and the CEM 170, a description of the IP address data is typically included. In some embodiments, data name/key, type, and length descriptions are included with the actual IP address data. In other embodiments, there the key directory server reduces the amount of descriptive information being sent. Every key in the directory server has a type and a length. Fields can be identified as variable length. Therefore, data type information need not be transmitted between elements in the system 100 if the elements use a common reference key stored in the directory server. Returning to the IP address data, by using the key directory server, elements need only send two bytes for the key id and four bytes for the actual address. Most of the data being sent in the system is relatively short in length. Therefore, the directory server helps reduce the amount of information being sent between the elements in the system 100.

Keys can be added to the directory server. The directory server can therefore support expansion of the kinds of fields being sent by allowing system

elements to update their locally stored key ids. For example, after a recipient receives a record with an "unknown" key, it contacts the directory server to get the key definition.

Central Database

The central database 175 is the optional central repository of the information collected by the system 100. The central database 175 is but one example of a sink for the data generated in the system 100. Other embodiments include other configurations. The central database 175 stores and maintains the data collected by the gatherers, as well as the information on the configuration of the system 100. Thus, in configuring the system 100, the NSP defines what data will be stored in each field in the central database 175 and how that data is collected from the ISMs.

The information on network sessions is stored in the database in the form of a table. Each field in the table represents a network session parameter. Each record describes a network session. The system 100 has a set of pre-defined fields that are configured by the CEM 170 on installation. The NSP can modify the central database 175 structure by adding, deleting, or modifying fields. The NSP access the data in the central database 175 by running queries and reports. The old data is removed from the central database 175 to free space for new data periodically. You can specify the time interval for which records are stored in the central database 175. The structure of the central database 175 with some of the predefined fields is illustrated in the following figure.

As each IP session may generate multiple transaction records, during the merge process the CEM 170 identifies and discards duplications, enhancing the efficiency of the data repository. Generally, data records are passed through the merger program, in the CEM 170, into the central database 175. However, the data records are also cached so that if matching records appear at some point, the already stored records can be replaced or enhanced with the new records. The database tables that contain the record flows can be indexed, enhancing the efficiency of the data repository. A merge is achieved by matching some of the fields in a data record and then merging the matching records from at least two record flows, transforming them into one record before updating the central database 175. In some embodiments, adaptive tolerance is used to match records. Adaptive tolerance allows for a variation in the values of fields that are compared (e.g., the time field value may be allowed to differ by some amount, but still be considered a match). The adaptive aspect of the matching can include learning the appropriate period to allow for the tolerance. The reason that the records that do not match any previous records are sent through into the central database 175, in addition to being cached for later matching, is to avoid loss of data in case of system failure.

The following table illustrates an example of the types of records stored in the central database 175 by the CEM 170.

Source IP	Destination IP	Source Host	Destination Host	Service	Date/Time	Duration	Total Bytes	Counter
199.203.13 2.187	204.71.177.3 5	pcLev.xacc t.com	yahoo.com	http	1998-04-26 10:56:55	6464	435666	261019
199.203.13	207.68.137.5	prodigy.xac	microsoft.co	telnet	1998-04-26	747	66743	261020

2.131	9	ct.com	m		10:56:55			
199.203.13 2.177	199.203.132. 1	pcEitan.xac ct.com	xpert.com	smtp	1998-04-26 10:56:55	82	55667	261021
199.203.13 2.173	204.162.80.1 82	pcAdi.xacc t.com	cnet.com	http	1998-04-26 10:56:55	93	33567	261022

The system 100 supports a non-proprietary database format enabling the central database 175 to run on any of a number of commercially available databases (e.g., MS-SQL Server , Oracle Server, DB2, etc.).

User Interface Server and Clients

The User Interface Server (UIS) 185 allows multiple clients (e.g. terminals 180) to access the system 100 through, the Microsoft Internet Explorer with Java™ Plug-in or Netscape Navigator with Java™ Plug-in. Other embodiments can use other applications to access the system 100. The main function of the UIS 185 is to provide remote and local platform independent control for the system 100. The UIS 185 can provide these functions through windows that correspond to the various components of the system 100. Access to the system 100 can be password protected, allowing only authorized users to log in to the system and protecting sensitive information.

The NSP can perform one or more of the following main tasks through the UIS 185:

- Configure the system 100.
- Create and run queries and reports on network activity and resource consumption.

- Register and license the system 100.

C. Data Distillation

Figure 2 illustrates the data distillation process performed by the system of Figure 1. The data distillation aggregates and correlate information from many different network devices to compile data useful in billing and network accounting.

First, the ISMs 210 gather data from their corresponding network device. Note that for some ISMs (e.g. pipe ISMs), real-time, policy-based filtering and aggregation 215 can also be done. This data is then fed to the gatherers 220. The gatherers 220 perform data enhancement to complete the data from the ISMs 210. The results are provided to the CEM 170. The CEM 170 performs data merges 270 to remove redundant data. The merged data is then optionally stored in the central database 175 as a billing record 275 or is sent directly to an external system. The billing record information can be accessed from external applications, through the application interface 290, via a data record 280. Filtering and/aggregation and/or data enhancements can be done at any stage in the system 100.

D. Data Enhancement

As mentioned above, the gatherers 220 provide data enhancement features to complete information received from the ISMs 210. The following

describes some example data enhancement techniques used in some embodiments of the invention.

Figure 3 illustrates an example of data enhancement. Data enhancement comprises a number of field enhancements. A field enhancement specifies how the data obtained from the trigger of the enhancement procedure is processed before it is placed in a single field in the central database 175. The data can be placed in the field directly, or new information may be added to the record by applying a Synchronous ISM function. (In the example below, the function is "resolve the IP address to a host FQDN"). Field enhancements may involve one or multiple steps. There is no limit to the number of steps in a Field Enhancement. The data record starts with fields obtained from an asynchronous ISM 300. The fields in the DR 300 are then enhanced using the field enhancements. The enhanced fields result in the DR 320.

A visual representation of an enhancement can be presented to the NSP. The enhancement may include an itinerary of ISMs starting off with an AISM, passing through PISMs, and terminating in the CEM 170. Using this view of the system 100, the NSP need not be shown the actual flow of data since the flow may be optimized later in order to achieve better performance. This is more of a graphical logical view of how the enhancement is achieved in steps. (PISMs can terminate more than one flow and initiate more than one flow.)

A visual representation of a field enhancement shows the per-field flow of data correlation. This process ends in the CEM 170 or in a PISM. The NSP

supplies information telling the system 100 how to reach each of the terminating fields (in the CEM 170 or the PISM) starting off from the initiating fields (PISM or AISM). Each step of enhancement defines cross correlation with some SISM function.

Figure 4A illustrates various field enhancements (410 through 440). A field enhancement includes applying zero or more functions to a field before storing the field in a specified field in the central database 175.

One-step Field Enhancement 410. The initial source data from the asynchronous ISM is placed directly in a field in the central database 175. Example: the field enhancement for the Source IP field.

Two-step Field Enhancement 420. The initial source data from the asynchronous ISM is used to obtain new additional data from a synchronous network device and the new data is placed in a field in the central database 175. Example: the field enhancement for the Source Host field.

Three-step Enhancement 430. The initial source data from the asynchronous ISM is used to obtain additional data from a synchronous ISM. The result is used to obtain more data from another ISM and the result is placed in a field in the central database 175.

The following illustrates an example data enhancement. Suppose the data obtained from a proxy server 101 contains the source IP address of a given session, such as 199.203.132.2, but not the complete domain address of the host computer (its Fully Qualified Domain Name), such as www.xacct.com. The

name of the host can be obtained by another network device - the Domain Name System (DNS 102) server. The DNS server 102 contains information that matches IP addresses of host computers to their Fully Qualified Domain Names (FQDNs). Through an enhancement procedure the information collected from the proxy server 101 can be supplemented by the information from the DNS 102. Therefore, the name of the host is added to the data (the data record) collected from the proxy server 101. The process of adding new data to the data record from different network devices can be repeated several times until all required data is collected and the data record is placed in the central database 175.

Figure 4B illustrates another example data enhancement where an enhanced record 490 is created from an initial netflow record 492. Fields in the enhanced record 490 are enhanced from the radius record 494, the QoS policy server record 496, the NMS DB record 498, and the LDAP record 499.

Defining Enhancement Procedures

The following describes the process for defining enhancement procedures in some embodiments of the system. Typically defining an enhancement procedures for the system 100 includes (1) defining enhancement procedures for each asynchronous ISM and (2) configuring field enhancements for all fields in the central database 175 for which the NSP wants to collect data originating from an asynchronous ISM that triggers the corresponding enhancement procedure.

An enhancement procedure can be defined as follows:

1. Access the CEM 170 using the UIS 180.
2. Select the enhancement procedures list using the UIS 180.
3. Define the name of the new enhancement procedure.
4. Select a trigger for the new enhancement procedure. The trigger can correspond to any asynchronous ISM in the system 100. Alternatively, the trigger can correspond to any asynchronous ISM in the system 100 that has not already been assigned to an enhancement procedure.
5. Optionally, a description for the enhancement procedure can be provided.
6. The new enhancement procedure can then be automatically populated with the existing fields in the central database 175. Optionally, the NSP can define the fields (which could then be propagated to the central database 175). Alternatively, based upon the type of asynchronous ISM, a preset set of fields could be proposed to the NSP for editing. What is important is that the NSP can define field procedures to enhance the data being put into the data records of the central database 175.
7. The NSP can then define the field enhancements for every field in the new enhancement procedure for which the NSP wants to collect

data from the ISM that is the trigger of the new enhancement procedure.

Defining Field Enhancements

Defining a field enhancement involves specifying the set of rules used to fill a database field from the information obtained from the trigger of the enhancement procedure. The NSP defines field enhancements for each field in which NSP wants to collect data from the trigger. If no field enhancements are defined, no data from the trigger will be collected in the fields. For example, suppose the firewall asynchronous ISM 130 that triggers an enhancement procedure. Suppose the central database 175 has the following fields: source IP, source host, destination IP, destination host, user name, total bytes, service, date/time, and URL. If the NSP wants to collect session data for each field except the URL from the firewall ISM 130, which triggers the enhancement procedure, the NSP defines a field enhancement for each field with the exception of the URL.

In some embodiments, the field enhancements are part of the enhancement procedure and the NSP can only define and modify them when the enhancement procedure is not enabled.

The field enhancements can be defined in a field enhancement configuration dialog box. The field enhancement configuration dialog box can have two panes. The first displays the name of the enhancement procedure, the name of its trigger, and the name and data type of the field for which the NSP is

defining the field enhancement. The second is dynamic and interactive. Its content changes depending on the NSP's input. When first displayed, it has two toggle buttons, End and Continue, and a list next to them. The content of the list depends on the button depressed.

When End is depressed, the list contains all output fields whose data type matches the data type of the field for which the NSP is defining the field enhancement. For example, if the field's data type is IP Address, the list contains all fields that are of the same type, such as source IP and destination IP that the AISM supplies. The fields in the list can come from two sources: (1) the source data which the gatherer receives from the trigger and (2) the result obtained by applying a synchronous ISM function as a preceding step in the field enhancement. The following notation is used for the fields:

OutputFieldName for the output of a field origination from the trigger

SISName.FunctionName(InputArgument).OutputField for the output of a field that is the result of applying a function

SISName...OutputField for the output of a field that is the result of applying a function as the final step of a field enhancement

The following examples are presented.

Source IP is the field provided by the trigger of the enhancement procedure that contains the IP address of the source host.

DNS...Host Name and DNS.Name(Source IP).Host name are the names of a field originating from the resolved function Name of a network device

called DNS that resolves the IP address to a domain address. The input argument of the function is the field provided by the trigger of the enhancement procedure, called source IP. It contains the IP address of the source host. The function returns the output field called Host Name that contains the domain address of the source host. The notation DNS...Host Name is used when the field is the result of applying the function as the final step of a field enhancement. The notation is DNS.Name(Source IP).Host Name is used when the field is used as the input to another function.

In the user interface, if End is unavailable, none of the output fields matches the data type of the field.

When Continue is depressed, the list contains all applicable functions of the available synchronous network device configured in the system 100. If the preceding output does not match the input to a function, it cannot be applied and does not appear on the list.

The following notation is used for the functions:

SISName.FunctionName(InputFieldName:InputFieldDataType)

→ *(OutputFieldName:OutputFieldDataType)*

When the function has multiple input and/or output arguments, the notation reflects this. The arguments are separated by commas.

The following example shows a field enhancement.

DNS. Address(Host Name:String) → (IP Address:IP Address)

Where DNS is the name of the synchronous ISM (or network device) as it appears in the system configuration.

Address is the name of the function.

(Host Name:String) is the input to the function - host FQDN of data type String

(IP Address:IP Address) is the output - IP address of data type IP Address

The NSP can define the field enhancement by choosing items from the list. The list contains the option <none> when the End button is depressed. Choosing this option has the same effect as not defining a field enhancement: no data from the trigger will be stored in the field in the central database 175.

At your request, here is one paragraph describing the algorithm(s) transforming the user representation of an Enhancement Procedure to an actual Computation Flow performed by the system. The source code accompanying this actually implements most of this. Of course, that code relies on the rest of the software (lots of other code), that may be required to completely understand it.

Field Enhancement Computation

In some embodiments, an enhancement procedure (EP) defines a flow of data records in the system 100, originating from an AISM that serves as a trigger, and terminating at the a database table, or a PISM (Pipe ISM - has the behavior of the database of being assigned input keys in much the same way,

but being installable/replacable logic in the system after it has been deployed), known as the Target of the EP. An EP is represented to the user as a collection of Field Enhancements (FEs). Each FE identifies what is performed in order to obtain a value (or not, if it is not defined), for that Field (either database field, or input key to PISM), using values originating from the Trigger AISM, possibly, utilizing functions of SISMs available in the system. The NSP defines every FE independently of other FEs, and completely defines what value is placed in the defined field.

In contrast to the above description of the presentation to the user of an EP, the system 100 (e.g., the CEM 170) computes what is called a Computation Flow, which is the actual operational instructions and parameters that are instructed to the gatherer's to achieve the desired NSP result (as defined in the EP). The Computation Flow is the detailed instructions required to achieve a "snowball" affect, of having data originating from a Trigger (AISM) in one gatherer , undergo specific enhancements by adding results from invocations of functions of SISMs with specific input parameters on the same gatherer or other gatherers (as required), and removing unused fields along this same path, eventually storing the user-configured results in the user-configured fields in an optimized fashion. The optimizations applied are:

- Transfer minimal data between any elements in the system;

- Do not compute the same function on the same input more than once. In other embodiments, this optimization may be changed to allow for sharing of intermediary values of field enhancements;
- Perform local enhancements before going on to a remote gatherer;
- Apply a "cost" function to transferring along communication links and traverse the minimal cost in a computation flow required to achieve correct results.

This set of optimizations can be performed, in some embodiments, using the code in Appendix A. That is, the transformation (from EP to Computation Flow, or simply Flow) is achieved by coupling together eleven algorithms and data transformations one after the other. These are described in the code module attached.

Note: 1) Within the code, EP, Rule, Computation Flow are used in some cases to refer to the same objects. 2) The eleven transformations that are applied to the user representation of the EP, after being stored in a database structure, are outlined beginning at line 124 of the attached source module "FlowManager.java".

E. Record Merges

Figure 5 illustrates an example record merge. Record merging removes duplicate records from the central database 175.

The following example shows how merges work and illustrates the need for merging duplicate records. Suppose the system 100 is using two

asynchronous ISMs 110 and 130. All outbound network traffic going through the proxy server 101 is routed through the firewall 103. The firewall 103 records the proxy server 101 as the source of all sessions passing through the proxy server 101, although they originate from different workstations on the network. At the same time, the proxy server 101 records the destination of all sessions as the firewall 103, although their actual destinations are the different Internet sites.

Therefore, all sessions are logged twice by the system 100 and the records are skewed. The data from the firewall 103 indicates the destination of a given session, but not the source (see data record 520), while the data from the proxy server 101 records the source, but not the destination (see data record 510). Defining a merge eliminates the duplication of records.

A merge can be defined instructing the CEM 170 to store the destination data obtained from the firewall 103 and the source data from the proxy server 101 in the central database 175. The merge will also eliminate the problem of skewed data by storing the correct source and destination of the session in the central database 175. Both network devices provide information on the URL. The latter can be used to identify the fact that the two seemingly independent records (510 and 520) are actually two logs of the same session.

Two enhancement procedures are defined for the example of Figure 5. The trigger of the first, designated Flow One, is the Proxy Server Asynchronous Information Source Module. The trigger of the second, Flow Two, is the

Firewall Asynchronous Information Source Module. The records from Flow One and Flow Two are records of the same session. They both have the same value for the URL field. Based on this value, the CEM 170 identifies the two records are double logs of the same session. It merges the two data records taking the Source IP value from Flow One and the Destination IP from Flow Two as the values to be stored in the central database 175.

Defining Merges

The following describes defining merges. A merge is a set of rules that specify how duplicate records from multiple enhancement procedures must be identified and combined before being stored in the central database 175. The NSP can merge the records from two or more enhancement procedures. To define a merge, the NSP identifies the following information.

- The enhancement procedures included in the merge.
- How to identify duplicate records (which fields of the records must match).
- How to combine the records; that is, for each field, which value (from which enhancement procedure) must be stored in the central database 175. (Optional)

If the NSP does not specify how records must be combined, the records are merged as follows:

- When the values in all but one of the fields are null, the non-null value is stored.

- When the fields contain non-null values, the value of the first record received (chronologically) is stored.

F. Additional Embodiments

The following describes additional embodiments of the invention.

In some embodiments, the user interface used by an NSP to configure the system 100 can be presented as a graphical representation of the data enhancement process. Every step in the enhancement can be shown as a block joined to another block (or icon or some graphical representation). The properties of a block define the operations within the block. In some embodiments, the entire data enhancement process from network devices to the central database 175 can be shown by linked graphics where the properties of a graphic are the properties of the enhancement at that stage.

In some embodiments, multiple CEMs 170 and/or central databases 175 can be used as data sources (back ends) for datamart or other databases or applications (e.g., customer care and billing systems).

In some embodiments, the types of databases used are not necessarily relational. Object databases or other databases can be used.

In some embodiments, other platforms are used. Although the above description of the system 100 has been IP network focussed with Unix or Windows NT systems supporting the elements, other networks (non-IP networks) and computer platforms can be used. What is important is that some

sort of processing and storing capability is available at the gatherers, the CEMs, the databases, and the user interface servers.

In some embodiments, the gatherers and other elements of the system 100, can be remotely configured, while in other embodiments, some of the elements need to be configured directly. For example, a gatherer may not be remotely configurable, in which case, the NSP must interface directly with the computer running the gatherer.

In other embodiments, the general ideas described herein can be applied to other distributed data enhancement problems. For example, some embodiments of the invention could be used to perform data source extraction and data preparation for data warehousing applications. The gatherers would interface with ISMs that are designed to extract data from databases (or other data sources). The gatherers would perform filtering and aggregation depending upon the needs of the datamart (in such an embodiment, the central database and CEM could be replaced with/used with a datamart). The data enhancement would then be done before storing the information in the datamart.

Figure 6 illustrates a system 600 where multiple systems 100 are linked together. This system could be an ISPs point of presence accounting system. The system 620 and the system 610 can store detailed network accounting information in their local detailed accounting databases. This information can then be aggregated and sent over the more expensive long distance links to the billing database in the system 630. Customer service information can still be

accessed at the detailed accounting database, but the aggregated information may be all that is needed to create the bills.

G. Conclusions

A network accounting and billing system and method has been described. In some embodiments, the system can access any network related information sources such as traffic statistics provided by routers and switching hubs as well as application server access logs. These are accumulated in a central database for creating auditing, accounting and billing reports. Because of the distributed architecture, filtering and enhancements, the system efficiently and accurately collects the network usage information for storage in a form that is useful for billing and accounting.

Appendix A - FlowManager Example

```
//*************************************************************************
//
// NAME & TITLE:
//  FlowManager  Analyzes the various Enhancemnt Rules and creates Computation
Flows
//
// ABSTRACT:
// -  Analyzes the various Enhancemnt Rules and creates Computation Flows
// -  Respond to Gatherer Requests for Instructions
//    Instructions are stored in UNIRInstruction, which can either be an element
of a
//    vector or a vector of UNIRInstructions in itself
// -  Allow for ReComputation of Particular Flows
// -  Implement Helper methods for finding interdependencies between Enhancemnt
Rules
//    and ISs that are referenced by them. Important when attempting to delete
an IS
//    or when an IS configuration has changed, to see which Enhancemnts mat be
//    affectetd.
//
// this class is closely related to the Gatherer.EnhanceCompFlowStep class,
which acts
// on the instructions created here.
// An instruction needs to know the following:
// 1. What type it is (recieve, enhance, send or other)
// 2. If it is of type enhance:
//      2.1 Where are the Input Unirs for the Enhancement Function
//          ("where" means position in the Arguments)
//      2.2 What IS and what function to perform
//      2.3 Where in the Arguments to put the function results.
// 3. If it is of type send, needs to know to whom to send. CEM or another
Gatherer.
//
// For further understanding, see Gatherer.EnhanceCompFlowStep
//
// CHANGE HISTORY:
//
//      Programmer     Change Date    Change Description
//      -------------- -----------    ---------------------------------------
//      Adi Gavish            01/12/97             Initial version
//
//*************************************************************************
import java.util.*;
```

```
import Util.*;
import Util.Persistence.*;
import Util.UNIRTypes.*;
import Util.Structs.*;
import Util.Log.*;
import COM.objectspace.jgl.*;

//*****************************************************************************
public class FlowManager
{
        // Current Rule we are processing
        int m_RuleID ;

        // Pipe ISM variables
        boolean m_IsTargetPipe = false ;
        int m_TargetIS       = 0 ;
        int m_TargetNode     = 1 ;
        int m_TargetGatherer = 0;

        // Holds all Enhancments for a Rule. Alll elements are of type
EnhanceOpNode.
        public EnhanceOpSList m_OpList  = new EnhanceOpSList();
        public EnhanceOpSList m_OpTmpList  = new EnhanceOpSList();

        // All output parameters are added to this list, with their positions in
the list
        // used to define the actual position of UNIRs in the Data Arguments
collected by
        // the ISs of the flow.
        // After Output positions are defined, the input parameters are updated
with
        // the new positions (So the Enhancement step in the Gatherer will know
where to
        // get the input parameters of some enhancement function)
        // see allocateUNIROutputData method
        SList UNIRAllocation ;

        // Temporary Storage of instructions converted from Enhancements
        SList m_TmpInstructions ;

        // HashTable For Flow Instruction Storage, access by RuleID Stores a
complete set
        // of instructions for every Flow (== Every Rule)
        Hashtable m_Instructions = new Hashtable();

        // Offset to fix UNIR Indexes (Take FlowID, Step into account)
        // See PlaceHolders in Gatherer.EnhanceCompFlowStep
        final static int FLOW_OFFSET = 2;

        // Globol counter used in DFS Topology Sort
```

```
    int m_TimeCtr = 0 ;

//*************************************************************************
    // Build All Flows
    // - Loop on all rules
    // - build an Instrucion vector for each Rule
    // - Store set of instructions per Rule in m_Instructions
//*************************************************************************
    public void computeFlows()
    {
    Rules RuleList = CEM.instance().getRules();
    if (RuleList != null)
    {
        // Loop on all Rules
          for(Enumeration El = RuleList.elements(); El.hasMoreElements(); )
          {
                    // Compute Flow for specific Rule (If Enhancement is
Enabled)
                  Rule CurrentRule = ( (Rule)(El.nextElement()) );
                  if (( CurrentRule != null ) && (CurrentRule.getEnabled()
== true) )
                  {
                      int RuleID = CurrentRule.getRuleID();
                        computeRuleFlows(RuleID);
                  }
          }
    }
    }

//*************************************************************************
    // Compute Flow for RuleID. At the end of this method, All
FlowInstructions for RuleID will be
    // generated and saved as an SList in a Hash Table, accessed by RuleID
    //
    // parameters
    //  RuleID - ID of rule to build the instruction set for
//*************************************************************************
    public void computeRuleFlows(int RuleID)
    {
    Log.instance().writeLog(new LogDebug("Compute Flow for Rule "+RuleID));
        m_RuleID =  RuleID ;

        try
        {
                // 1. Build Initial EnhanceOpNode set for RuleID
                buildGUONSByRule(RuleID);
                if ( m_OpList.size() > 0)
                {
                        // 2. Create Enhancement String Descriptions
                            buildOpDescriptions();
```

```
                                        // 3. Convert All Opnums in Enhancement Input Params
to GUONS
                                        buildTranslatedOpKeyIn();
                                        // 4. Reduce Duplicate GUONS (with same Description)
                                        reduceSameDescriptionOps();
                                        // 5a. Generate OpKeysOut Information
                                            buildOpKeysOut();
                                        // 5b. Reduce GUONS that are NOT INPUT to any other
Non End GUON.
                                        //      Dependent on 5a.
                                        reduceErrorGuons();
                                        // 6a. Perform Topology Sorting
                                    doTopolgySort();
                                        // 6b. Perform "Node Coupling" Optimizations
                                    doNodeCoupling();
                                        // 7. Allocate UNIR Data Structure Key Index
                                        allocateUNIROutputData();
                                        // 8. For every input parameter, match the appropriate
Index fronm step 7
                                        propogateIndexToInput();
                                    }
                                // 9. Create IntructionSet
                                    buildInstructionSet();
                                // 10. Save IntructionSet in Hash Table with RuleID as Key
                                m_Instructions.put(new Integer(RuleID), m_TmpInstructions);
                                // 11. Build FieldIndex Vector & send to Merger (if Target is
DB, not pipe)
                                if (!m_IsTargetPipe)
                                {
                                        buildFieldIndex();
                                    }
                            Log.instance().writeLog("FlowManager.computeRuleFlows",
LogEvent.LOG_DEBUG,
                                                "Done Compute Flow for Rule "+RuleID );
                        }
                        catch (Exception E)
                        {
                                Debug.writeStackTrace("FlowManager.computeRuleFlows", E);
                        }
                }

        //**************************************************************************
                // Build an initial set of EnhanceOpNodes representing all Enhancements
per some
                // rule. Set some additional Info on every EnhanceOpNode, such as
GathererID and
                // m_IsLast Flag.
                // !!!! IT IS ASSUMED THAT Enhancements (From PERSISTENCE) is loaded
sorted by
                // Field/OpNum !!!
```

```
//**********************************************************************
    protected void buildGUONSByRule( int RuleID )
{
    m_IsTargetPipe = false ;
    m_TargetNode     = 1 ;
    m_TargetGatherer = 0 ;
    m_TargetIS       = 0 ;

    m_OpList = new EnhanceOpSList ();
    Nodes NodesList = CEM.instance().getNodes();
          ISs   ISsList    = CEM.instance().getISs();

    // Loop on All Enhancements. For Every Enhancement that matches the
current
    // Rule, create an EnhanceOpNode object and add it to the OpArray. After
    // loop is done, Sort the OpArray.
    // OpArray will sort on ?
    Enhancements EnhancementList = CEM.instance().getEnhancements();
    Array OpArray = new Array();
    if (EnhancementList != null)
    {
                  for(Enumeration E = EnhancementList.elements();
E.hasMoreElements(); )
          {
                  Enhancement EnhancementItem = (
(Enhancement)(E.nextElement()) );
                  if (EnhancementItem.getRuleID() == RuleID)
                  {
               //printDiagnosticsPre(Enhancement);
               EnhanceOpNode EnhancementNode = new EnhanceOpNode(
EnhancementItem );
               //printDiagnosticsPost(EnhancementNode);
                  OpArray.add( EnhancementNode );
          }
          }
    }
          Sorting.sort( OpArray );

    // build m_OpList from OpArray
    if (OpArray.size() > 0)
    {
          String LastField = "";
          Rules RuleList = CEM.instance().getRules();
          Rule CurRule = RuleList.getRuleByRuleID(RuleID);
          // If Rule Not Enabled, m_OpList will be empty
          if  ( (CurRule != null) && (CurRule.getEnabled()) )
          {
              // Get the AISM ID of this Flow
              int ISID = CurRule.getTriggerID();
```

```
                    // Set Target Pipe Boolean, Target Node and other Pipe
variables
                      m_TargetIS = CurRule.getTargetID();
                      if (m_TargetIS > 0)
                      {
                          m_IsTargetPipe = true ;
                                  IS TargetIS = ISsList.getISByISID(
m_TargetIS );
                                  if (TargetIS != null)
                                  {
                                          m_TargetGatherer =
TargetIS.getGathererID();
                              Node N =
NodesList.getNodeByTypeID(Util.Comm.Node.NODE_TYPE_GATHERER, m_TargetGatherer);
                                  m_TargetNode = N.getNodeID();
                                  }
                      }

                    // Get AISM Information ()
                              IS I = ISsList.getISByISID( ISID );
                              if (I != null)
                              {
                    int ctr = 0 ;
                                  // Build First entry into m_OpList from thr
Rule Header
                                  int GathererID = I.getGathererID();
                                  int ISMID = I.getISMID();
                          Node N =
NodesList.getNodeByTypeID(Util.Comm.Node.NODE_TYPE_GATHERER, GathererID);
                          int NodeID = N.getNodeID();

                          // Add GUON 0 with data from the Rule Header
                                  EnhanceOpNode OpZero = new EnhanceOpNode
(RuleID, "", (short)0, ISID, 0, null, null);
                          OpZero.setGathererID(GathererID);
                          OpZero.setNodeID(NodeID);
                          OpZero.setISMID(ISMID);
                          // setting GUON to the original index will garuentee
unique value.
                                  OpZero.setGUON(ctr);
                              OpZero.setLastFlag(false) ;
                          m_OpList.add ( OpZero );
                          ctr++;

                          // Loop on OpArray, build the rest of m_OpList
                                  EnhanceOpNode PriorEnhanceOp = null ;
                                  for(Enumeration E = OpArray.elements();
E.hasMoreElements(); )
                              {
```

```
                                    EnhanceOpNode CurrentEnhancement = (
(EnhanceOpNode)(E.nextElement()) );
                                    // Check if it belongs to The Rule
                                    if ( ( CurrentEnhancement != null ) &&
(CurrentEnhancement.getRuleID() == RuleID ) )
                                    {
                                        ctr++;
                                            // set aux data, add to list
                                        I = ISsList.getISByISID(
CurrentEnhancement.getISID() );
                                        if (I!=null)
                                        {
                                            GathererID = I.getGathererID();
                                            ISMID = I.getISMID();
                                            N =
NodesList.getNodeByTypeID(Util.Comm.Node.NODE_TYPE_GATHERER, GathererID);
                                            NodeID = N.getNodeID();


CurrentEnhancement.setGathererID(GathererID);

CurrentEnhancement.setNodeID(NodeID);

CurrentEnhancement.setISMID(ISMID);
                                 // set GUON to the original index will garuentee
unique value.

CurrentEnhancement.setGUON(ctr);

    CurrentEnhancement.setLastFlag(false) ;

                                            // Check if Field changed. If
Field has changed, this means that
                                            // the previous EnhanceOpNode
was the last for that field.
                                            String CurrentField =
CurrentEnhancement.getFieldID();
                                            if ( (
!CurrentField.equals(LastField) ) && (PriorEnhanceOp != null) )
                                            {
                                                // Field Has Changed, prior
Op was last for a Field

    PriorEnhanceOp.setLastFlag(true) ;
                                                // Set Target Field of
previous EnhanceOpNode

    PriorEnhanceOp.addOutFields(LastField);
                                            }
```

```
                                        // Save for Last Step Per Field
Logic
                                        PriorEnhanceOp =
CurrentEnhancement ;
                                        LastField =  CurrentField ;

                                        // Add To List
                                        m_OpList.add (
CurrentEnhancement );

        Log.instance().writeLog("FlowManager.buildGUONSByRule",
LogEvent.LOG_DEBUG, CurrentEnhancement.toString() );

                                    }
                                    else
                                    {
                                            Log.instance().writeLog(new
LogError("buildGUONSByRule: No IS found for "+CurrentEnhancement.getISID() ));
                                    }
                                }
                            }
                            // End Of Loop Logic. Same as if Field has
changed.
                            if (PriorEnhanceOp != null)
                            {
                                PriorEnhanceOp.setLastFlag(true) ;
                                PriorEnhanceOp.addOutFields(LastField);
                            }
                        }
                    }
                }
            }

    //***************************************************************************
    // Print Diagnostics on the passed Enhancement.
    // Currently Not Used.
    //***************************************************************************
        protected void        printDiagnosticsPre(Enhancement EnhancementItem)
        {
            try
            {
                byte[] buf = EnhancementItem.getOpKeyIn();
                String InputList = "[" ;
            Arguments a = new Arguments ( buf );
            for (Enumeration EK = a.elements(); EK.hasMoreElements(); )
            {
                short value =
((UNIRShort)EK.nextElement()).getShortValue();
                InputList  += Short.toString(value)+", ";
```

```
                        value       =
((UNIRShort)EK.nextElement()).getShortValue();
                        InputList  += Short.toString(value)+", ";
                }
                InputList += "]";
                        Log.instance().writeLog("FlowManager.buildGUONSByRule",
                                LogEvent.LOG_DEBUG, "PersistenceItem:
"+InputList);

                }
                catch(Exception e)
                {
                }
        }

    //*********************************************************************
    // Print Diagnostics on the passed EnhanceOpNode.
    // Currently Not Used.
    //*********************************************************************
        protected void       printDiagnosticsPost(EnhanceOpNode
EnhancementNode)
        {
                try
                {
                        byte[] buf = EnhancementNode.getOpKeyIn();
                        String InputList = "[" ;
                Arguments a = new Arguments ( buf );
                for (Enumeration EK = a.elements(); EK.hasMoreElements(); )
                {
                        short value =
((UNIRShort)EK.nextElement()).getShortValue();
                        InputList  += Short.toString(value)+", ";
                        value       =
((UNIRShort)EK.nextElement()).getShortValue();
                        InputList  += Short.toString(value)+", ";
                }
                InputList += "]";
                        Log.instance().writeLog("FlowManager.buildGUONSByRule",
                                LogEvent.LOG_DEBUG, "Struct 1:
"+InputList);

                        InputList = "[" ;
                for (Enumeration EK =
EnhancementNode.getParsedOpKeyIn().elements(); EK.hasMoreElements(); )
                {

                        int value    = ( (Integer)EK.nextElement() ).intValue();
                        InputList  += Integer.toString(value)+", ";
                        value        = ( (Integer)EK.nextElement() ).intValue();
                        InputList  += Integer.toString(value)+", ";
```

```
            }
            InputList += "]";
                  Log.instance().writeLog("FlowManager.buildGUONSByRule",
                                  LogEvent.LOG_DEBUG, "Struct 2:
"+InputList);


            }
            catch(Exception e)
            {
            }
      }

   //**********************************************************************
   // ABSTRACT
      //  Convert the m_OpList to an Instruction Set the Gatherer Can
Understand
      //  (a set of UNIRInstructions)
      //
      // Description
      //  Translate m_OpList to sets of Enhancement Instructions. Divide the
Enhancements
      //  by Different Gatherer sequences, wrap each segment with a Receive
Instruction
      //  and a Send Instruction.
   //**********************************************************************

            //***************************************************
            // Helper method to add an instruction of type Send
            //***************************************************
                  void AddSendStep(int StepCtr, int NodeID, int ISID, int
ISMID, int GathererID)
                  {
                  UNIRInstruction u = new UNIRInstruction (m_RuleID,

                   StepCtr,

UNIRInstruction.INSTRUCTION_SEND,
                                                  NodeID,
                                                  0, ISID, ISMID,
                                                  null, null,
GathererID );
                  Log.instance().writeLog("FlowManager.buildInstructionSet",
LogEvent.LOG_DEBUG,
                                    "Adding Instruction "+u);
                        m_TmpInstructions.pushBack(u);

                  }

            //***************************************************
```

```
// Helper method to add an receive of type Receive
//**************************************************
        void AddReceiveStep(int StepCtr, int NodeID, int ISID, int
ISMID, int GathererID)
        {
        UNIRInstruction u = new UNIRInstruction (m_RuleID,

         StepCtr,

UNIRInstruction.INSTRUCTION_RECEIVE,
                                                  NodeID,
                                                  0, ISID, ISMID,
                                                  null, null,
GathererID );
        Log.instance().writeLog("FlowManager.buildInstructionSet",
LogEvent.LOG_DEBUG,
                                "Adding Instruction "+u);
            m_TmpInstructions.pushBack(u);

        }

//**************************************************
// Helper method to add a receive of type Enhance
//**************************************************
        void AddEnhanceStep(int StepCtr, int NodeID, int ISID, int
ISMID, int GathererID,
                            int FunctionID, byte[] inParam, byte[]
outParam)
        {
        UNIRInstruction u = new UNIRInstruction (m_RuleID,

         StepCtr,

UNIRInstruction.INSTRUCTION_ENHANCE,
                                                  NodeID,
                                                  FunctionID, ISID,
ISMID,

            inParam, outParam, GathererID);
        Log.instance().writeLog("FlowManager.buildInstructionSet",
LogEvent.LOG_DEBUG,
                                "Adding Instruction "+u);
            m_TmpInstructions.pushBack(u);

        }

//**************************************************
// Helper method to add an receive of type Pipe Give
//**************************************************
```

```
                    void AddGiveStep(int StepCtr, int NodeID, int ISID, int
ISMID, int GathererID,
                                    int FunctionID, byte[] inParam, byte[]
outParam)
                    {
                    UNIRInstruction u = new UNIRInstruction (m_RuleID,

                     StepCtr,

UNIRInstruction.INSTRUCTION_GIVE,
                                                            NodeID,
                                                            FunctionID, ISID,
ISMID,

                         inParam, outParam, GathererID);
                    Log.instance().writeLog("FlowManager.buildInstructionSet",
LogEvent.LOG_DEBUG,
                                        "Adding Instruction "+u);
                        m_TmpInstructions.pushBack(u);

                    }

            //************************************************

        protected void      buildInstructionSet()
        {
        Log.instance().writeLog("FlowManager.buildInstructionSet",
LogEvent.LOG_DEBUG,
                                    "Build Instruction Set for Rule
"+m_RuleID);
            EnhanceOpNode PriorOp = null ;
            int PriorNodeID = -1;
            m_TmpInstructions =  new SList ();
            int StepCtr = 0;
        for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
        {
            EnhanceOpNode Op1 = (EnhanceOpNode)I.get();
            if ( !Op1.getLastFlag() )
            {
                if (Op1.getNodeID() != PriorNodeID)
                {
                    // Add Send Instruction to close prior node
sequence
                    if (PriorNodeID != -1)
                    {
                                AddSendStep(StepCtr,
Op1.getNodeID(), Op1.getISID(),
                                                        Op1.getISMID(),
PriorOp.getGathererID());
                            StepCtr++;
```

```
                                }
                                // Add Recieve Instruction to start current node
sequence
                                AddReceiveStep(StepCtr, Op1.getNodeID(),
Op1.getISID(),
                                                Op1.getISMID(),
Op1.getGathererID());
                                    StepCtr++;
                                PriorNodeID = Op1.getNodeID();
                            }
                                PriorOp = Op1 ;
                            // Add Enhancement Step
                                AddEnhanceStep(StepCtr, Op1.getNodeID(),
Op1.getISID(), Op1.getISMID(),
                                                Op1.getGathererID(),
Op1.getFunctionID(),

        Op1.getOpKeyInAsArray(), Op1.getOpKeyOutAsArray());
                                StepCtr++;
                            }
        }
        //  The Last Step, Add Send Instruction to CEM
        //  Check globol Pipe flag. If this Flow Target is some Pipe ISM,
        //  Set the Instructione Type to INSTRUCTION_GIVE and set correct Node
        if (PriorOp != null)
        {
            if (m_IsTargetPipe)
            {
                // if target is in a different node, add a send and recieve
                if (m_TargetNode != PriorOp.getNodeID())
                {
                                    AddSendStep(StepCtr, m_TargetNode, 0, 0,
PriorOp.getGathererID());
                                    StepCtr++;
                                AddReceiveStep(StepCtr, 0, 0, 0,
m_TargetGatherer);
                                    StepCtr++;
                }
                // adding the final Give instruction. Dont forget Binding
Information
                                byte[] bindingInfo =
buildPISMBindingInformation();
                                AddGiveStep(StepCtr, m_TargetNode, m_TargetIS, 0,
                                            m_TargetGatherer, 0,
bindingInfo, null);
                            }
                else
            {
                // Send Data to CEM
```

```
                        AddSendStep(StepCtr, Util.Comm.Node.NODE_TYPE_CEM,
0, 0, PriorOp.getGathererID());
            }
                StepCtr++;
            }
        }

    //*************************************************************************
        // doTopolgySort()
        // m_OpList can be seen as a Directed Acyclic Graph (DAG), and an
algorithm
        // (Introduction To Algorithms, Chapter 23) to sort the graph is applied.
The
        // result of this procedure is a list where the order of operations is
correct,
        // meaning that all enhancments whose ouput is used as input happen
before the
        // enhancments that use them.
    //*************************************************************************

                protected void visitDFS(EnhanceOpNode Op)
                {
                        Op.setColor(EnhanceOpNode.GRAY);
                        Op.setDFSDiscoverTime(m_TimeCtr);
                        m_TimeCtr++;
                        // For Each Adjacent Vertex (All Output GUONS)
                for (Enumeration AdjVert = Op.getOutputGUONs();
AdjVert.hasMoreElements(); )
                {
                        int OutGuon =
((Integer)AdjVert.nextElement()).intValue();
                        EnhanceOpNode OutOp = m_OpList.findGuon(OutGuon);
                        // Explore Edges Each (Guon,OutGuon) pair is an
edge
                        if ( ( OutOp != null) && (OutOp.getColor() ==
EnhanceOpNode.WHITE) )
                        {
                                visitDFS(OutOp);
                                OutOp.setPredecessor(Op);
                        }
                }
                Op.setColor(EnhanceOpNode.BLACK);
                        Op.setDFSFinishTime(m_TimeCtr);
                        m_TimeCtr++;

                        m_OpTmpList.pushFront(Op);
                }

                //-------
```

```
protected void doTopolgySort()
{
        // Create Temprary List to Store Ordered Ops.
m_OpTmpList = new EnhanceOpSList ();

        m_TimeCtr = 0 ;
        // For each vertex
for (int i=0;i<m_OpList.size();i++)
{
        EnhanceOpNode Op1 = (EnhanceOpNode)m_OpList.at(i);
        // if not used in DFS yet (Color White)
        if ( Op1.getColor() == EnhanceOpNode.WHITE)
        {
                visitDFS( Op1 );
        }
}

// Temporary List is now the correct one, move it to normal list
m_OpList =  new EnhanceOpSList ( m_OpTmpList );
}

//*******************************************************************
*************************
// doNodeCoupling()
// Node Coupling attempts to move enhancments that share the same
gatherer closer
// in sequence, as a way to reduce traffic between gatherers. This is
done without
// damaging the basic order of the  enhancements established earlier in
the
// Topology Sort.
//*******************************************************************
*************************

                //----------------------
                // Find if inserting Op  at Pos will cause sort order
conflict
                protected boolean noHarmToSort(EnhanceOpSList OpList, int
pos, EnhanceOpNode Op)
                {
                        for (int i=pos;i<OpList.size();i++)
                        {
                        EnhanceOpNode OpToCompare =
(EnhanceOpNode)OpList.at(i);
                                // Check if Op has input that uses Output
of OpToCompare
                        // Loop on all inputs of Op, see if the Guon value
matches OpToCompare.Guon
                        for (Enumeration EK = Op.getInIterator();
EK.hasMoreElements(); )
```

```
                    {
                            OpInputParam InParam =
(OpInputParam)EK.nextElement();
                                    if ( InParam.getGUON() ==
OpToCompare.getGUON())
                                    {
                                            // No insertion Possible
                                            return false ;
                                    }
                    }
                    }
                    // No Conflict Found, Insertion Allowed
                    return true ;
            }

            //----------------------
            // find last position of first same node sequence in
OpList
            protected int findSameNodePos(EnhanceOpSList OpList,
EnhanceOpNode Op)
            {
                    int NodeID = Op.getNodeID();
                    boolean StartSeqFound = false ;
                    int ctr = 0;
                    int ResultVal = -1;
            for (SListIterator I = OpList.begin(); (!I.atEnd());
I.advance())
            {
                    EnhanceOpNode OpToCompare =
(EnhanceOpNode)I.get();
                    if (NodeID == OpToCompare.getNodeID())
                    {
                            StartSeqFound = true ;
                    }
                    else
                    {
                            if ( StartSeqFound )
                            {
                                    ResultVal = ctr ;
                                    break; // get out of loop, we found
what weere looking for
                            }
                    }
                    ctr++;
            }

                    return ResultVal ;
            }

            //-------
```

```
protected void doNodeCoupling()
    {
            // Create Temprary List to Store Ordered Ops.
    m_OpTmpList = new EnhanceOpSList ();

    for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
    {
            EnhanceOpNode Op1 = (EnhanceOpNode)I.get();
            // Add Op1 to the Temporary List, either at the end (No Node
Optimization Possible)
            // or Insert in the middle (Node Optimization, Couple Nodes)
            int n1 = findSameNodePos(m_OpTmpList, Op1);
            if (n1 == -1)
            {
                    // No Node Optimizations possible, add to end
                    m_OpTmpList.pushBack(Op1);
            }
            else
            {
                    if ( noHarmToSort(m_OpTmpList, n1, Op1) )
                    {
                            // Insertion OK, insert to group same node Ops
together
                            m_OpTmpList.insert(n1, Op1);
                    }
                    else
                    {
                            // Insertion would hurt the input->output order,
cannot inset, add to end
                            m_OpTmpList.pushBack(Op1);
                    }
            }
    }

    // Temporary List is now the correct one, move it to normal list
    m_OpList =  new EnhanceOpSList ( m_OpTmpList );
    }

    //*********************************************************************
********
    // Create an image of the Data to be produced by this Rule. This
allocates ouput
    // positions for every Enhancement. These positions are used later to
update the
    // input params.
    //*********************************************************************
********
    protected void allocateUNIROutputData()
    {
```

```
			UNIRAllocation = new SList();
			// For Each OP
		for (int i=0;i<m_OpList.size();i++)
		{
			EnhanceOpNode Op1 = (EnhanceOpNode)m_OpList.at(i);
			if (Op1 != null)
			{
				// For Each Output Parameter
				for (Enumeration EK = Op1.getOutputKeys();
EK.hasMoreElements(); )
				{
					Integer Key = (Integer)EK.nextElement(); // first
is key, 2nd is guon
					GuonKeyParam NewKey = new GuonKeyParam(Key, new
Integer(Op1.getGUON() ) );
					if (NewKey!=null)
					{
						UNIRAllocation.pushBack(NewKey);
					}
					}
				}
			}
		}

		//*******************************************************************
********
		// propogateIndexToInput()
		// Based on the Unir Allocation, match every input parameter with the
index in the
		// UNIRAllocation.
		//*******************************************************************
********

				protected int findIndexinAllocation ( OpInputParam KeyIn)
				{
					GuonKeyParam CurInKey = KeyIn.getGuonKey();
					for (int i=0;i<UNIRAllocation.size();i++)
					{
						GuonKeyParam KeyInAllocation =
(GuonKeyParam)UNIRAllocation.at(i);
						if (CurInKey.equals(KeyInAllocation))
						{
							return i ;
						}
					}
					return -1 ;
				}

			//---
```

```
protected void propogateIndexToInput()
{
        // For Each OP
for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
{
        EnhanceOpNode Op1 = (EnhanceOpNode)I.get();
        // For Each Input Parameter
        for (Enumeration EK = Op1.getInIterator(); EK.hasMoreElements();
)
        {
                OpInputParam InParam = (OpInputParam)EK.nextElement();
                // Get index  of pair in UNIRAllocation
                int n = findIndexinAllocation(InParam);
                if (n == -1)
                {
                        // TBD, Error
                }
                InParam.setIndex(n + FLOW_OFFSET );
        }
        Log.instance().writeLog("FlowManager.propogateIndexToInput",
LogEvent.LOG_DEBUG,
                                "Propagated Input Op: " +Op1);
}
}

//**************************************************************************
********
// For every Enhancement
//   Build List of Enhancments that need the output of some enhancement
as Input.
//
// This information is needed later to weed out Non productive
enhancements and
// to perform the Topological sort on m_OpList
//**************************************************************************
********
protected void buildOpKeysOut()
{
    // Op1 Loop : Loop on All Enhancements in m_OpList
for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
{
        EnhanceOpNode Op1 = (EnhanceOpNode)I.get();
    // Op2 Loop :  Loop on All Enhancements in m_OpList
    // Compare all Enhancments to Op1 (Except itself)
        for (SListIterator J = m_OpList.begin(); (!J.atEnd());
J.advance())
        {
                EnhanceOpNode Op2 = (EnhanceOpNode)J.get();
                // No need to compare Op to itself
                if (Op1!=Op2)
```

```
                                        {
                                            // If Op1 participates as input in Op2, add Op2 to
list of
                                            // Enhancements that need Op1 output. This list is
managed in Op1.
                                                for (Enumeration EK = Op2.getInIterator();
EK.hasMoreElements(); )
                                                {
                                                        OpInputParam InParam =
(OpInputParam)EK.nextElement();
                                                        if (InParam.getGUON() == Op1.getGUON())
                                                        {

        Op1.addOutputParam(InParam.getKey(), Op2.getGUON());
                                                        }
                                                }
                                            }

                                }
                        }
                }

        //*******************************************************************
********
        // Reduce Duplicates. No need for duplication of process if another
enhancement
        // already creates the needed Output. Keep one of the duplicates, delete
the other.
        // Pass thru all input keys of all enhancments, replacing any reference
to the
        // Deleted GUON with the Surviving GUON.
        //*******************************************************************
********
        protected void reduceSameDescriptionOps()
        {
        for (int i=0;i<m_OpList.size();i++)
        {
                EnhanceOpNode Op1 = (EnhanceOpNode)m_OpList.at(i);
                // Move from end to Current Pos because we might delete an object
                for (int j=m_OpList.size()-1;j>i;j--)
                {
                        EnhanceOpNode Op2 = (EnhanceOpNode)m_OpList.at(j);
                        if ( Op1.getDescription().equals( Op2.getDescription() ) )
                        {

        Log.instance().writeLog("FlowManager.reduceSameDescriptionOps",
LogEvent.LOG_DEBUG,
                                                "Reducing Description :
"+Op2.getFieldID()+":"+
                                                                Op2.getGUON()+":"+
```

```
Op2.getDescription() );
                                        // Merge outfields of deleted guon with surviving
guon
                                        Op1.addOutFields(Op2.getOutFields() );
                                        int newGUON = Op1.getGUON();
                                        int oldGUON = Op2.getGUON();
                                        m_OpList.remove(j);
                                        // GUON Replacement Pass
                                        for (int k=0;k<m_OpList.size();k++)
                                        {
                                                EnhanceOpNode Op3 =
(EnhanceOpNode)m_OpList.at(k);
                                                Op3.SwapInputKeyGUON(oldGUON, newGUON);
                                        }
                                        // Mark for Garbage Collection
                                        Op2 = null ;
                                }
                        }
                }
        }

        //**********************************************************************
********
        // Reduce Error Guons, that do not produce an output needed as input by
any other
        // guon. They can be identified if they are not "Last" and the output
list is empty.
        //**********************************************************************
********
        protected void reduceErrorGuons()
        {
                if (m_OpList.size() > 0)
                {
                for (int i=m_OpList.size()-1;i>=0;i--)
                {
                        EnhanceOpNode Op1 = (EnhanceOpNode)m_OpList.at(i);
                        // If Not Last Guon per some field...
                        if (!Op1.getLastFlag())
                        {
                                // Check if any Outputs are defined
                                        Enumeration ChkOutput =
Op1.getOutputGUONKeys();
                                        if (!ChkOutput.hasMoreElements())
                                        {
                        Log.instance().writeLog("FlowManager.reduceErrorGuons",
LogEvent.LOG_DEBUG,
                                                "Reducing Bad Enhancment
:"+Op1.getGUON()+":"+Op1.getDescription());
```

```
                                        // No Output Elements for this
Guon, delete it
                                m_OpList.remove(i);
                                // Mark for Garbage Collection
                                Op1 = null ;
                                }
                        }
                }
            }
    }

        //*************************************************************************
********
        // Build Translated OpKeyIn for every Enhancment
        // !!!! IT IS ASSUMED THAT Enhancements (From PERSISTENCE) are loaded
sorted by
        // Field/Opnum !!!
        //*************************************************************************
********
        protected void buildTranslatedOpKeyIn()
    {
        for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
        {
                EnhanceOpNode Op = (EnhanceOpNode)I.get();
                if (Op != null)
                {
                        Op.translateOpKeyIn(m_OpList);

        Log.instance().writeLog("FlowManager.buildTranslatedOpKeyIn",
LogEvent.LOG_DEBUG, Op.toString() );
                }
        }
    }

        //*************************************************************************
********
        // Build a Description for every Enhancment
        // !!!! IT IS ASSUMED THAT Enhancements (From PERSISTENCE) is loaded
sorted by
        // Field/Opnum !!!
        //*************************************************************************
********
        protected void buildOpDescriptions()
    {
        for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
        {
                EnhanceOpNode Op = (EnhanceOpNode)I.get();
                if (Op != null)
                {
                        String S = getDescription( Op );
```

```
        Log.instance().writeLog("FlowManager.buildOpDescriptions",
LogEvent.LOG_DEBUG,
                                                    "Op Description :
"+Op.getFieldID()+":"+Op.getGUON()+": " +S);
                }
            }
    }

        //*******************************************************************
********
        // get Textual representation of the first EnhanceOpNode Input Key
        //*******************************************************************
********
        protected String getKeyText(EnhanceOpNode Op)
        {
                String ResultVal = "" ;
                try
                {
                        if (Op != null)
                        {
                                Arguments args = new Arguments( Op.getOpKeyIn() );
                                int n =
((UNIRShort)args.get(ArgKeys.KEY_KEY_ID)).getShortValue();
                                ResultVal = Integer.toString(n);
                        }
                }
                catch (Exception e) {}
                return ResultVal ;
        }

        //*******************************************************************
********
        // Build a Textual description of a regular EnhanceOpNode Op
        //
        // the methods "parseArgs" and "getDescription" are used in a recursive
manner
        //*******************************************************************
********
        protected String parseArgs(EnhanceOpNode Op)
        {
                String s = "" ;
         try
         {
                // Loop On All Op Key In
                Arguments a = new Arguments ( Op.getOpKeyIn() );
                for (Enumeration EK = a.elements(); EK.hasMoreElements(); )
                {
                        // Add Description  of Input Param OpNum
```

```
                        short OpValue  = ((UNIRShort)
EK.nextElement()).getShortValue();
                        EnhanceOpNode CurrentOp = m_OpList.findOp(OpValue,
Op.getFieldID() );
                        if ( !Op.equals(CurrentOp) )
                        {
                                s += getDescription(CurrentOp)+".";
                                // Add Description  of Input Param Key
                                short KeyValue = ((UNIRShort)
EK.nextElement()).getShortValue();
                                s += Integer.toString(KeyValue);
                                // If More Param, delimit by comma
                                if ( EK.hasMoreElements() )
                                {
                                        s+="," ;
                                }
                        }
                        else
                        {
        Log.instance().writeLog("FlowManager.parseArgs", LogEvent.LOG_ERROR,
"Corruption Detected in Data (Illegal Recursion)");
                                throw ( new Exception() );
                        }
                }
            }
            catch(Exception EA)
            {
                s = "" ;
            }
            return s ;
        }

        //*********************************************************************
********
        // Build a Textual description of the EnhanceOpNode Op
        //
        // the methods "parseArgs" and "getDescription" are used in a recursive
manner
        //*********************************************************************
********
        protected String getDescription(EnhanceOpNode Op)
        {
                String s = Op.getDescription();
                // If No description was built yet, build it and save for next
time
                if (s==null)
                {
                    // if last Step for some field, just parse
                        if (Op.getLastFlag())
```

```
                        {
                                s = parseArgs(Op);
                        }
                        else
                                if (Op.getOpNum() == 0)
                                {
                                        // First Op, has no function
                                        s = Op.getISID() + "." + getKeyText(Op);
                                }
                                else
                                {
                                        // Function, Must show Func ID + Input
Parameters.
                                        s = Op.getISID() + "." + Op.getFunctionID()
+ "(";
                                        s += parseArgs(Op);
                                    s += ")" ;
                                }
                        //----  Save for next time
                    Op.setDescription(s);
                        //----
                }
                return s ;
                }

        //*************************************************************************
*
        // Scan All Rules, Loop On All Instructions, send matching instructions
to Gatherer
        //*************************************************************************
********
    public void setInstructionsByGatherer( int GathererID, GathererController GC
)
    {
        Rules RuleList = CEM.instance().getRules();
        if (RuleList != null)
        {
            // For Each Rule
                for(Enumeration E1 = RuleList.elements(); E1.hasMoreElements(); )
                {
                        // Build Flow for specific Rule
                    Rule CurrentRule = ( (Rule)(E1.nextElement()) );
                    if (( CurrentRule != null ) && (CurrentRule.getEnabled()
== true) )
                    {
                        int RuleID = CurrentRule.getRuleID();
                            setInstructionsByRuleByGatherer( GathererID,
RuleID, GC );
                    }
                }
```

```
        }
    }

        //***********************************************************************
*
        // Send All Instructions per Rule/ per Gatherer to the Gatherer
        // Thru the GathererController
        //***********************************************************************
*
    synchronized public void setInstructionsByRuleByGatherer( int GathererID,
int RuleID, GathererController GC )
    {
                SList TmpInstructions = (SList)m_Instructions.get(new
Integer(RuleID));
                Vector v = new Vector(); // Hold UNIRInstruction belonging to
some Gatherer
                // Loop on All instructions of a Rule, Add Gatherer Matches to v,
Send v to GC
                for (Enumeration EOps = TmpInstructions.elements();
EOps.hasMoreElements();)
                {
                UNIRInstruction Op = (UNIRInstruction)EOps.nextElement();
                if (Op.getGathererID() == GathererID)
                {
                        v.addElement(Op);
                }
                }
                try
                {

        Log.instance().writeLog("FlowManager.setInstructionsByRuleByGatherer",
LogEvent.LOG_DEBUG, "Disable Flow "+RuleID+ " on " + GathererID);
                        GC.disableFlow(RuleID);
                        if (v.size() > 0)
                        {

        Log.instance().writeLog("FlowManager.setInstructionsByRuleByGatherer",
LogEvent.LOG_DEBUG, "Upload Flow "+RuleID+ " on " + GathererID);
                                GC.sendFlowData(RuleID, new UNIRInstruction(v) );

        Log.instance().writeLog("FlowManager.setInstructionsByRuleByGatherer",
LogEvent.LOG_DEBUG, "Enable Flow "+RuleID+ " on " + GathererID);
                                GC.enableFlow(RuleID);
                        }
                }
                catch(Exception e)
                {
                }
        }
```

```
//**********************************************************************
*
// Pack the Binding Information between current Flow to the input
// parameters of the target Pipe ISM.
// This information is added to the Give instruction and sent to the
// Gatherer where it is used to pass data from the output UNIR of
// the current flow to the input parameters of the target flow.
// RETURNS
// a byte array containing Binding information in the following format:
// pairs of shorts made of the index of the data in the output UNIR,
followed
// by the key value of the target input parameter.
//**********************************************************************
*
byte[] buildPISMBindingInformation()
{
        try
        {
                Arguments args = new Arguments();
                Vector v = new Vector();
                // Loop on All Enhancements
        for (SListIterator I = m_OpList.begin(); (!I.atEnd());
I.advance())
        {
                EnhanceOpNode Op = (EnhanceOpNode)I.get();
                if (Op != null)
                        {
                                int    Index   = 0; // Position of data in
Output Unir
                                short  Key     = 0; // Key ID   of data in
Output Unir
                                int inKey      = 0; // Key ID   of data in
Input Parameters
                        // the binding process will
                        // "take the UNIR in the Index position from the
flow data and put
                        // in the PISM input parameter in inKey"

                        // One to many relation:  One input to Many output
                        // DEFINE INPUT : Get Index & key of first input
key
                        Enumeration EK = Op.getInIterator();
                        if ( EK.hasMoreElements() )
                        {
                                OpInputParam InParam =
(OpInputParam)EK.nextElement();
                                        Index = InParam.m_KeyIndex ;
                                        Key   = (short)InParam.m_KeyID;

                                }
```

```
                        // DEFINE OUTPUT(S) Loop on all Out Values
                            for (Enumeration EFlds =
Op.getOutFields().elements(); EFlds.hasMoreElements();)
                            {
                                String FieldID =
(String)EFlds.nextElement();

        Log.instance().writeLog("FlowManager.buildPISMBindingInformation",
LogEvent.LOG_INFO,
                                                              "Got Pipe
Binding: "+ FieldID +" ; "+ Index +" ; "+ Key);
                                // in PISM, FieldID is a String
Representation of the KeyID
                                // defined as string for
compatabiliy with Fields (see buildFieldIndex)
                            inKey  = Integer.valueOf( FieldID
).intValue();

                            // Add to Arguments;

        args.put(ArgKeys.ARGKEY_ENHANCEMENT_OP_NUM, new UNIRShort((short)Index));
                                args.put(ArgKeys.KEY_KEY_ID, new
UNIRShort((short)inKey));

                            }
                    }
                }
                return args.toByteArray();
        }
        catch(Exception e)
        {

        Log.instance().writeLog("FlowManager.buildPISMBindingInformation",
LogEvent.LOG_ERROR,
                                                             "Error in
Binding Info, "+e);

        Debug.writeStackTrace("FlowManager.buildPISMBindingInformation", e);


        }
        return null ;
        }

        //*********************************************************************

        // Builds a vector of all DB fields that participate in a specific Flow.
        // (The current flow, in m_OpList)
        // This list is built of  FieldIndexStruct elements and sent to the
Merger.
```

```
        // The Merger uses this information in processing incoming UNIRs from
this
        // flow.
        // The vector is a cross reference between the DB FieldID and the
position
        // the data to be put in the field from the incoming UNIR for this flow.
        //*************************************************************************
*
    void buildFieldIndex()
    {
                Vector v = new Vector();
                // Loop on All Enhancements
        for (SListIterator I = m_OpList.begin(); (!I.atEnd()); I.advance())
        {
                EnhanceOpNode Op = (EnhanceOpNode)I.get();
                if (Op != null)
                        {
                                int     Index   = 0;
                                short   Key     = 0;
                        Enumeration EK = Op.getInIterator();
                        // Get Index & key of first input key
                        if ( EK.hasMoreElements() )
                        {
                                OpInputParam InParam =
(OpInputParam)EK.nextElement();
                                        Index = InParam.m_KeyIndex ;
                                        Key   = (short)InParam.m_KeyID;

                                }
                        // Loop on all Fields Out, add all fields in list to
vector with
                        // Index & ket stored earlier.
                                for (Enumeration EFlds =
Op.getOutFields().elements(); EFlds.hasMoreElements();)
                                {
                                        String FieldID =
(String)EFlds.nextElement();
                                        v.addElement(new FieldIndexStruct(FieldID,
Index, Key));
                                }
                }
        }
        if (v.size() > 0)
        {
                        // Call Merger
                        Merger m = CEM.instance().getMerger();
                Log.instance().writeLog(new LogDebug("Vector:\n"+v));
                        m.addRule(m_RuleID, v);
                }
                else
```

```
            // Call Merger
            Merger m = CEM.instance().getMerger();
        Log.instance().writeLog(new LogDebug("Deleting Rule"+m_RuleID));
            m.deleteRule(m_RuleID);
        }
    }

    //***********************************************************************
*
    // Get All Enhancments with matching FieldID
    // Return Vector of afffected Rules
    //***********************************************************************
*
    public Vector getRulesWithField(String FieldID)
    {
        Log.instance().writeLog("FlowManager.getRulesWithRuleField",
LogEvent.LOG_DEBUG, "Get Enhancements with Field "+FieldID);
        Vector ResultVal = new Vector();
        OrderedSet AffectedRules = new OrderedSet(false);
        Vector ToDel = new Vector();
     Enhancements EnhancementList = CEM.instance().getEnhancements();
        for(Enumeration E = EnhancementList.elements();
E.hasMoreElements(); )
        {
            Enhancement CurrentEnhancement = (
(Enhancement)(E.nextElement()) );
        if ( ( FieldID.equals( CurrentEnhancement.getFieldID() ) ) )
        {
            AffectedRules.add( new Integer (
CurrentEnhancement.getRuleID() ) );
        }
    }

        // Set Result Val
        for(Enumeration AR = AffectedRules.elements();
AR.hasMoreElements(); )
        {
            Integer RuleIDObj = (Integer)AR.nextElement();
            ResultVal.addElement(RuleIDObj); // add Rule to Result Set
        }
    return ResultVal ;
    }

    //***********************************************************************
*
    // Delete All Enhancments with matching FieldID, RuleID from List
    //***********************************************************************
*
```

```
public void deleteRulesWithField(Vector RuleList, String FieldID)

{
        for (int i=0;i<RuleList.size();i++)
        {
                int RuleID =
((Integer)(RuleList.elementAt(i))).intValue();
                deleteRulesWithRuleField(RuleID, FieldID);
        }
}


//*****************************************************************
*
// Delete All Enhancments with matching RuleID, FieldID
// Return Vector of afffected Rules
//*****************************************************************
*
public Vector deleteRulesWithRuleField(int RuleID, String FieldID)

{
        Log.instance().writeLog("FlowManager.deleteRulesWithRuleField",
LogEvent.LOG_DEBUG, "Delete Enhancements with Field "+FieldID);
        Vector ResultVal = new Vector();
        OrderedSet AffectedRules = new OrderedSet(false);
        Vector ToDel = new Vector();
Enhancements EnhancementList = CEM.instance().getEnhancements();
        for(Enumeration E = EnhancementList.elements();
E.hasMoreElements(); )
        {
                Enhancement CurrentEnhancement = (
(Enhancement)(E.nextElement()) );
        if ( ( FieldID.equals( CurrentEnhancement.getFieldID() ) ) &&
             ( (RuleID == CurrentEnhancement.getRuleID() ) ) )
             //( (RuleID == CurrentEnhancement.getRuleID() ) || (RuleID
== -1) ) )
        {
                ToDel.addElement(CurrentEnhancement);
        }
}
for (int i=0;i<ToDel.size();i++)
{
        Enhancement CurrentEnhancement = (Enhancement)ToDel.elementAt(i);
        AffectedRules.add( new Integer ( CurrentEnhancement.getRuleID() )
);

Log.instance().writeLog("FlowManager.deleteRulesWithRuleField",
LogEvent.LOG_DEBUG, "Deleteing "+CurrentEnhancement);
        try { CurrentEnhancement.delete();} catch (Exception e){}
}
```

```
                    // Set Result Val
                    for(Enumeration AR = AffectedRules.elements();
AR.hasMoreElements(); )
                    {
                            Integer RuleIDObj = (Integer)AR.nextElement();
                            ResultVal.addElement(RuleIDObj); // add Rule to Result Set
                    }
            return ResultVal ;
            }


            //*********************************************************************
*
            // deleteRuleEnhancements
            //*********************************************************************
*
            public void deleteRuleEnhancements(int RuleID)
            {
                    Vector ToDel = new Vector();
            Enhancements EnhancementList = CEM.instance().getEnhancements();
                    for(Enumeration E = EnhancementList.elements();
E.hasMoreElements(); )
                    {
                            Enhancement CurrentEnhancement = (
(Enhancement)(E.nextElement()) );
                    if ( CurrentEnhancement.getRuleID() == RuleID )
                    {
                            ToDel.addElement(CurrentEnhancement);
                    }
                }
            for (int i=0;i<ToDel.size();i++)
            {
                    Enhancement CurrentEnhancement = (Enhancement)ToDel.elementAt(i);
                    try
                    {
                            CurrentEnhancement.delete();
                            }
                    catch (Exception e)
                    {
                            Log.instance().writeLog(new LogDebug("Delete Rule
Enhancement Failed... "+e));

            Debug.writeStackTrace("FlowManager.deleteRuleEnhancements", e);
                            }
            }
            }

            //*********************************************************************
*
```

```
 / Delete All Enhancments with matching ISID
// If All enhancements per rule are deleted, disable the Rule
// If Trigger is Deleted, Delete Rule
// Return Vector of afffected Rules
//**********************************************************************

public Vector deleteRulesWithIS(int ISID)
{
Log.instance().writeLog(new LogDebug("Delete Enhancements for "+ISID));
        Vector ResultVal = new Vector();
        OrderedSet AffectedRules      = new OrderedSet(false);
        OrderedSet AffectedRuleFields = new OrderedSet(false);
        Vector ToDel = new Vector();
        // Loop on all Rules
 Rules RuleList = CEM.instance().getRules();
        for(Enumeration E1 = RuleList.elements(); E1.hasMoreElements(); )
        {
            // Compare Trigger ID to the ISID being deleted
                Rule CurrentRule = ( (Rule)(E1.nextElement()) );
            int RuleID = CurrentRule.getRuleID();
            if (CurrentRule.getTriggerID() == ISID)
            {
                // yes. found a match. Add to Affected List
                // and delete the whole Flow
               AffectedRules.add( new Integer ( RuleID ) );
               ToDel.addElement(CurrentRule);
               deleteRuleEnhancements(RuleID);
            }
        }
        // Delete from Collection
for (int i=0;i<ToDel.size();i++)
{
        Log.instance().writeLog(new LogDebug("Delete Rule with Trigger
"+ISID));
        Rule CurrentRule = (Rule)ToDel.elementAt(i);
        try { CurrentRule.delete();} catch (Exception e){}
}


        // Loop on All Enhancements, find Rule/Feild pairs affected by IS
removal
  Enhancements EnhancementList = CEM.instance().getEnhancements();
        for(Enumeration E2 = EnhancementList.elements();
E2.hasMoreElements(); )
        {
                Enhancement CurrentEnhancement = (
(Enhancement)(E2.nextElement()) );
                if (CurrentEnhancement.getISID() == ISID)
                {
                Log.instance().writeLog(new LogDebug("Adding Rule Field to
Affected List "+
```

```
CurrentEnhancement.getRuleID()+" : "+

CurrentEnhancement.getFieldID() ));
                                        AffectedRuleFields.add(new Pair(new Integer
(CurrentEnhancement.getRuleID()),

CurrentEnhancement.getFieldID() ) );
                              }
                }

                    for(Enumeration E3 = AffectedRuleFields.elements();
E3.hasMoreElements(); )
                    {
                              Pair CurrentPair = ( (Pair)(E3.nextElement()) );
                              // Delete All Enhancemnts for this Rule & FieldID

          deleteRulesWithRuleField(((Integer)CurrentPair.first).intValue(),
(String)CurrentPair.second);
                    AffectedRules.add( (Integer)CurrentPair.first );
               }

                    // Set Result Val
                    for(Enumeration AR = AffectedRules.elements();
AR.hasMoreElements(); )
                    {
                              Integer RuleIDObj = (Integer)AR.nextElement();
                              ResultVal.addElement(RuleIDObj); // add Rule to Result Set
                    }

               // Return Affected Rules
          return ResultVal ;
          }

          //*****************************************************************************
*
          // Delete from the given Rule all the Fields that have Enhancements using
          // at least 1 from the given set of Trigger Input Keys .
          //*****************************************************************************
*
          public OrderedSet deleteRuleFieldsWithTriggerKeys (int RuleId, Vector
TriggerKeys )
          {
               Log.instance().writeLog(new LogDebug
("deleteRuleFieldsWithTriggerKeys "+"RuleId=" + RuleId+
                                        ", TriggerKeys="+ TriggerKeys));
               int OpNum = 0;
               OrderedSet AffectedRule = new OrderedSet(false);
               OrderedSet AffectedFields = new OrderedSet(false);
```

```
                if ( TriggerKeys==null || TriggerKeys.size()==0)
                    return null;
                Log.instance().writeLog(new LogDebug ("Add Rule to Affected List "+
RuleId));
                AffectedRule.add(new Integer(RuleId));

                //Get all Enhancements
                Enhancements EnhancementList = CEM.instance().getEnhancements();
            for(Enumeration E = EnhancementList.elements(); E.hasMoreElements(); )
            {
                Enhancement CurrentEnhancement = ( (Enhancement)(E.nextElement()) );
                if (CurrentEnhancement.getRuleID()==RuleId)
                // Search InputKeys for Trigger ones (OpNum = 0)
                {
                   EnhanceStruct CurrentEnhanceStruct = new
EnhanceStruct(CurrentEnhancement);
                   Vector ParsedOpKeyIn = CurrentEnhanceStruct.getParsedOpKeyIn();
                   ///
                   int CurrentOpNum;
                   int CurrentOpKey;
                   for (int i=0; i<ParsedOpKeyIn.size(); i+=2)
                   {
                       CurrentOpNum =
((Integer)ParsedOpKeyIn.elementAt(i)).intValue();
                       CurrentOpKey =
((Integer)ParsedOpKeyIn.elementAt(i+1)).intValue();
                       if ( CurrentOpNum==OpNum && TriggerKeys.contains (new
Integer(CurrentOpKey)) )
                       {
                           Log.instance().writeLog(new LogDebug("Adding Rule Field to
Affected List "+

CurrentEnhancement.getRuleID()+" : "+

CurrentEnhancement.getFieldID() ));
                           AffectedFields.add( CurrentEnhanceStruct.getFieldID());
                           break; //pass to next Enhancement
                       }
                   } // for over ParsedOpKeyIn
               }//if (CurrentEnhancement.getRuleID()==RuleId)

           }//loop over Enhancements
           for(Enumeration E = AffectedFields.elements(); E.hasMoreElements(); )
                   {
                       String FieldId = (String)E.nextElement();
                       // Delete All Enhancemnts for this Rule & FieldID
                           deleteRulesWithRuleField(RuleId, FieldId);
           }
           return AffectedRule;
           }//deleteRuleFieldsWithTriggerKeys
```

```
//**********************************************************************
// Delete in all Rules all Fields including an Enhancement from the given
// ISID with FunctionId  equal to one from the given set.
//**********************************************************************
public OrderedSet deleteRulesFieldsWithISFunctions (int ISID , Vector
ChangedFunctions)
{
    Log.instance().writeLog(new LogDebug
("deleteRulesFieldsWithISFunctions "+ "ISID="+ISID+
                    ", ChangedFunctions="+ChangedFunctions ));
    Vector ToDel= new Vector();
    OrderedSet AffectedRules = new OrderedSet(false);
    OrderedSet AffectedFields = new OrderedSet(false);

    if ( ChangedFunctions==null || ChangedFunctions.size()==0)
        return null;
    Enhancements EnhancementList = CEM.instance().getEnhancements();
for(Enumeration E = EnhancementList.elements(); E.hasMoreElements(); )
{
    Enhancement CurrentEnhancement = ( (Enhancement)(E.nextElement()) );
    if (CurrentEnhancement.getISID() == ISID  &&
ChangedFunctions.contains( new Integer(CurrentEnhancement.getFunctionID())) )
    {
        //ToDel.addElement(CurrentEnhancement);
        Log.instance().writeLog(new LogDebug("Adding Rule Field to
Affected List "+

CurrentEnhancement.getRuleID()+" : "+

CurrentEnhancement.getFieldID() ));
                        AffectedFields.add(new Pair(new Integer
(CurrentEnhancement.getRuleID()),

CurrentEnhancement.getFieldID() ) );
    }
}
for(Enumeration E2 = AffectedFields.elements(); E2.hasMoreElements(); )
      {
            Pair CurrentPair = ( (Pair)(E2.nextElement()) );
            // Delete All Enhancemnts for this Rule & FieldID

deleteRulesWithRuleField(((Integer)CurrentPair.first).intValue(),
(String)CurrentPair.second);
            Log.instance().writeLog(new LogDebug("Add Rule to Affected
List "+CurrentPair.first));
      AffectedRules.add( (Integer)CurrentPair.first );
    }

return AffectedRules;
```

```
·//deleteRulesFieldsWithISFunctions

//**********************************************************************
*
// In all Rules find all Enhancements which met the fllowing conditions:
//      ISID = <given ISID>, FunctionID = <one from the given list>.
// For all these Enhancements the FunctionId is replaced with new one .
// The "new" Id is placed immidiately after the matching "old" Id in the
// 2-nd input parameter.
//**********************************************************************
*
public OrderedSet editEnhancementsFunctionIDWithFunctionID(int ISID ,
Vector AffectedFunctionIds)
        throws Exception
    {
        Log.instance().writeLog(new LogDebug
("editEnhancementsFunctionIDWithFunctionID "+ "ISID="+ISID+
                    ", AffectedFunctionIds="+AffectedFunctionIds ));
        OrderedSet AffectedRuleSet = new OrderedSet(false);

        if ( AffectedFunctionIds==null || AffectedFunctionIds.size()==0)
            {
            Log.instance().writeLog(new LogDebug
("editEnhancementsFunctionIDWithFunctionID: ", "There are no changed function
IDs" ).;
            return null;
            }
        Enhancements EnhancementList = CEM.instance().getEnhancements();
    for(Enumeration E = EnhancementList.elements(); E.hasMoreElements(); )
    {
        Enhancement CurrentEnhancement = ( (Enhancement)(E.nextElement()) );
        for (int i=0;i<AffectedFunctionIds.size();i+=2)
        {
            if (CurrentEnhancement.getFunctionID()==
((Integer)AffectedFunctionIds.elementAt(i)).intValue())
            {
                int OldFuncId = CurrentEnhancement.getFunctionID();
                int NewFuncId =
((Integer)AffectedFunctionIds.elementAt(i)).intValue();
                Log.instance().writeLog(new LogDebug ("Replace FunctionId
"+OldFuncId +
                        "with " + NewFuncId+ " in Enhancement
="+CurrentEnhancement));
                CurrentEnhancement.edit();
                CurrentEnhancement.setFunctionID(NewFuncId);
                CurrentEnhancement.update();
                Log.instance().writeLog(new LogDebug("Add Rule to Affected
List "-CurrentEnhancement.getRuleID()));
                AffectedRuleSet.add (new Integer
(CurrentEnhancement.getRuleID()));
```

```
            }
        }

        return AffectedRuleSet;
        }
}//class

//***************************************************************************
// Internal Helper Classes to support Flow computations
//***************************************************************************

        //***********************************************************************
*
        //
        //***********************************************************************
*
    class EnhanceOpSList extends SList
    {
        //***********************************************************************
*
        //
        //***********************************************************************
*
        public EnhanceOpSList()
        {
                super();
        }


        //***********************************************************************
*
        //
        //***********************************************************************
*
        public EnhanceOpSList(int n)
        {
                super(n);
        }


        //***********************************************************************
*
        //
        //***********************************************************************
*
```

```
public EnhanceOpSList(EnhanceOpSList s)
{
        super(s);
}


//*****************************************************************
*
//

//*****************************************************************
*
        public EnhanceOpNode findOp (int OpValue, String OpField)
        {
        for (int i=0;i<size();i++)
        {
                EnhanceOpNode Op = (EnhanceOpNode)at(i);
                if (Op.getOpNum() == OpValue)
                {
                        // Field does not matter if we are searching for
OpNum 0
                        if ( (OpField.equals( Op.getFieldID() ) ) ||
(OpValue == 0) )
                        {
                                return Op ;
                        }
                }
        }
                return null ;
        }


//*****************************************************************
*
//

//*****************************************************************
*
        public EnhanceOpNode findGuon (int GuonValue)
        {
        for (int i=0;i<size();i++)
        {
                EnhanceOpNode Op = (EnhanceOpNode)at(i);
                if (Op.getGUON() == GuonValue)
                {
                        return Op ;
                }
        }
                return null ;
        }
```

```
}
//**********************************************************************
*

//**********************************************************************
*
// Object that represents a single Input Parameter of a single Enhancement
// Operation.
// Every Enhancment should maintain a set of OpInputParam
//**********************************************************************
*
class OpInputParam
{
    // Original Opnum of Enhancment owning the Input Param
    public int m_OpNum    = 0 ;
    // ArgKey Value of the Output Parameter
    public int m_KeyID = 0 ;
    // GUON value of Enhancment owning the Input Param
    public int m_OpGUON = 0 ;
    // Index of Input Parameter. Defined from the "UNIR Allocation"
    public int m_KeyIndex = 0 ;

    ////////////
    // Constructors
    ////////////
    public OpInputParam( int OpNum, int KeyID, int GUON)
    {
                m_OpNum     = OpNum ;
                m_KeyID     = KeyID ;
                m_OpGUON    = GUON ;
                m_KeyIndex = 0 ;
    }

    ////////////
    // Get Set
    ////////////
    public int getGUON()                        {return m_OpGUON;}
    public void setGUON(int Value)          {m_OpGUON = Value;}
    public int getKey()                             {return m_KeyID;}
    public void setKey(int Value)               {m_KeyID = Value;}

    public GuonKeyParam getGuonKey()        {return new GuonKeyParam(new Integer(m_KeyID), new Integer(m_OpGUON));}

    public int getIndex()                       {return m_KeyIndex;}
    public void setIndex(int Value)         {m_KeyIndex = Value;}
```

```
}
//***********************************************************************
*


//***********************************************************************
*

//
//***********************************************************************
*

class GuonKeyParam extends Pair
{
public int m_GUON = 0 ;
public int m_Key  = 0 ;

        public GuonKeyParam ()
        {
                super();
        }

        public GuonKeyParam (Integer Key, Integer GUON)
        {
                super (Key, GUON);
        }

        public int hashCode()
        {
                Integer n1 = (Integer)first;
                Integer n2 = (Integer)second;
                return ( (n1.intValue() << 16)+ n2.intValue() );
        }

        public boolean equals(Pair pair)
        {
                Integer n1 = (Integer)first;
                Integer n2 = (Integer)pair.first;
                Integer n3 = (Integer)second;
                Integer n4 = (Integer)pair.second;
                return ( ( n1.intValue() == n2.intValue() )&& (
n3.intValue() == n4.intValue() ) );
        }


}

//***********************************************************************
*
// Object that representes a single Output Parameter of a single
Enhancement Operation.
```

```
        // Every Enhancment should maintain a set of OpOutputParam
        //*********************************************************************
*
    class OpOutputParam
    {
        // Set of Key/GUON pairs, elements are JGL Pairs
        public OrderedSet m_KeyGUONList = new OrderedSet ( false ); // No
Duplicates
        // Set of Keys. elements are Integers
        public OrderedSet m_KeyList     = new OrderedSet ( false ); // No
Duplicates
        // Set of GUONs. elements are Integers
        public OrderedSet m_GUONList     = new OrderedSet ( false ); // No
Duplicates


        //*********************************************************************
*
        // Get Set
        //*********************************************************************
*
        public Enumeration getGUONs()        {return m_GUONList.elements();}
        public Enumeration getKeys()                 {return
m_KeyList.elements();}
        public Enumeration getGUONKeys()     {return m_KeyGUONList.elements();}

        //*********************************************************************
*
        // FFU:
        //public Keys getGUONKeys(GUON);
        //public GUONs getKeyGUONs(Key);


//*********************************************************************
            //

//*********************************************************************
                public void addKey(int Key, int GUON)
                {
                        Pair KeyGuon = new GuonKeyParam (new Integer(Key), new
Integer (GUON) ) ;
                        m_KeyGUONList.add(KeyGuon);
                        m_KeyList.add(new Integer(Key));
                        m_GUONList.add(new Integer(GUON));
                }


//*********************************************************************
            //
```

```
//*******************************************************************
            public byte[] getOpKeyOutAsArray()
            {
                  try
                  {
                  Arguments args = new Arguments();
                  for (Enumeration EK = getKeys(); EK.hasMoreElements(); )
                  {
                        Integer outKey = (Integer)EK.nextElement();
                              args.put(ArgKeys.KEY_KEY_ID, new
UNIRShort((short)outKey.intValue() ));

                  }
                  return args.toByteArray();
               }
               catch (Exception e){}

               return null ;
            }

//*******************************************************************

    }

      //*******************************************************************
*
      // Object that representes a single Enhancement Operation.
      // Flow Manager should maintain a set of EnhanceOpNode
      //*******************************************************************
*
   class EnhanceOpNode extends EnhanceStruct
   {
            // Unique Identifier of the Op, Created at Runtime. "Global
Unique Op Num"
      int    m_GUON = 0 ;
      // String Description of the Operation, Created "Recursively" at Runtime.
      String m_Description = null ;

            // OrderedSet version of EnhanceStruct.OpKeysIn. Each element is
of type OpInputParam
      OrderedSet  m_TranslatedOpKeysIn      = new OrderedSet ( false ); // No
Duplicates
      Vector      m_TranslatedOpKeysInVector = new Vector();

            // List of Fields that are the "End Product" of this GUON. null
in most objects, has value only for
            // "End" Objects. Becuase of Optimizations, one GUON can feed
more than one field. Elements are Strings
            // that were the FieldID of Original Enhancement.
```

```
OrderedSet m_OutFields = new OrderedSet( false ) ;

// Object that manages the lists of output param keys of this Enhancment
      OpOutputParam  m_OpKeysOut = new OpOutputParam();

// ISM, Gatherer & Node ID of this (derived from EnhanceStruct.ISID)
int m_ISMID      = 0;
int m_GathererID = 0;
int m_NodeID     = 0;

// Flag if this GUON was originally the "last" (Last OpNum for a FIELDID)
boolean m_IsLast = false ;

        // The list of EnhanceOpNodes can be viewed as a Directed Acyclic
Graph (DAG), with the GUONS as vertices
        // and the pairs of (GUON n, GUON m using output of n as Input)
as edges.
        // The Following are Variables Used in a TOPLOGICAL SORT using
Depth First Search (DFS).
        final static int WHITE = 0;
        final static int GRAY = 1;
        final static int BLACK = 2;
        public int m_ColorStatus = WHITE;
        public EnhanceOpNode m_Pi = null;     // Predecessor Node. if
null, then root (Several roots possible)
        public int m_DiscoverTime = 0 ;      // Start Time of DFS on this
GUON
        public int m_FinishTime = 0;                // End   Time of DFS
on this GUON (and adjaceny list)

////////////
// Constructors
////////////
public EnhanceOpNode( Enhancement E)
{
    super (E);
}

public EnhanceOpNode(int RuleID, String FieldID, short OpNum, int ISID,
int FunctionID, byte[] OpKeyIn,
                                                Vector ParsedOpKeyIn
)
{
    super (RuleID, FieldID, OpNum, ISID, FunctionID, OpKeyIn,
ParsedOpKeyIn );
}

////////////
// Operations
////////////
```

```
//***********************************************************************
*
            // Create a list of OpInputParam in m_TranslatedOpKeysIn from the
pair
            // arguments in m_OpKeyIn. Add the corresponding GUON for every
OpNum
            // in the OpKeyIn.
//***********************************************************************
*
            public void translateOpKeyIn(EnhanceOpSList OpList)
            {
                try
                {
                Arguments a = new Arguments ( getOpKeyIn() );
                for (Enumeration EK = a.elements(); EK.hasMoreElements();
)
                {
                    short OpValue  = ((UNIRShort)
EK.nextElement()).getShortValue();
                    short KeyValue = ((UNIRShort)
EK.nextElement()).getShortValue();
                    OpInputParam tmp = null ;
                    EnhanceOpNode CurrentOp = OpList.findOp(OpValue,
m_FieldID);
                    tmp = new OpInputParam (OpValue, KeyValue,
CurrentOp.getGUON() );
                    m_TranslatedOpKeysIn.add(tmp);
                    m_TranslatedOpKeysInVector.addElement(tmp);
                }
              }catch(Exception EA){}
            ;

//***********************************************************************
*
            // pass thru m_TranslatedOpKeysIn, replacing occurences of
oldGUON with newGUON
//***********************************************************************
*
            public void SwapInputKeyGUON (int oldGUON, int newGUON)
            {
            for (Enumeration EK = m_TranslatedOpKeysIn.elements();
EK.hasMoreElements(); )
            {
                OpInputParam InParam = (OpInputParam)EK.nextElement();
                if (InParam.getGUON() == oldGUON)
```

```
                {
                        InParam.setGUON(newGUON);
                }
            }
        };

//***********************************************************************
*
        // Add Op to Lists of output of current EnhanceOpNode
//***********************************************************************
*
        public void addOutputParam    (int Key, int GUON)
        {
                m_OpKeysOut.addKey(Key, GUON);
        }

//***********************************************************************
*
// Get Set
//***********************************************************************
*
public int getGUON()                                    {return m_GUON;}
public void setGUON(int Value)                    {m_GUON = Value;}
public int getGathererID()                                      {return
m_GathererID;}
public void setGathererID(int Value)              {m_GathererID = Value;}
public int getNodeID()                                     {return
m_NodeID;}
public void setNodeID(int Value)                     {m_NodeID = Value;}
public boolean getLastFlag()                      {return m_IsLast ;}
public void setLastFlag(boolean value)                {m_IsLast = value ;}
public String getDescription()                    {return m_Description ;}
public void setDescription(String value)          {m_Description = value ;}
public int getISMID()                                      {return
m_ISMID;}
public void setISMID(int Value)                            {m_ISMID =
Value;}

//***********************************************************************
*
// Topology Sort
//***********************************************************************
*
```

```
        public int getColor()                                    {return
m_ColorStatus;}
        public void setColor(int Value)                     {m_ColorStatus =
Value;}
        public void setDFSDiscoverTime(int Value)           {m_DiscoverTime =
Value;}
        public void setDFSFinishTime(int Value)             {m_FinishTime =
Value;}
                public void setPredecessor(EnhanceOpNode Value)     {m_Pi=Value;}


        //*********************************************************************
*
                public void addOutFields(String FieldID)
        {m_OutFields.add(FieldID);}
                public void addOutFields(OrderedSet FieldList)      {m_OutFields
= m_OutFields.union(FieldList);}
                public OrderedSet getOutFields()                    {return
m_OutFields;}
                public Enumeration getInIterator()          {return
m_TranslatedOpKeysInVector.elements();}
        public Enumeration getOutputGUONs()                 {return
m_OpKeysOut.getGUONs();}
        public Enumeration getOutputGUONKeys()                      {return
m_OpKeysOut.getGUONKeys();}
        public Enumeration getOutputKeys()                          {return
m_OpKeysOut.getKeys();}



        //*********************************************************************
*
                // Convert OpKeyIn structure to a byte array for the
UNIRInstruction

        //*********************************************************************
*
            public byte[] getOpKeyInAsArray()
            {
                try
                {
                        Arguments args = new Arguments();
                        for (Enumeration EK = getInIterator();
EK.hasMoreElements(); )
                        {
                                OpInputParam InParam =
(OpInputParam)EK.nextElement();
                                        args.put(ArgKeys.ARGKEY_ENHANCEMENT_OP_NUM,
new UNIRShort((short)InParam.m_KeyIndex));
```

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☑ OTHER: small text

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.